QAS-Bench: Rethinking Quantum Architecture Search and A Benchmark

Xudong Lu [1] **Kaisen Pan** [1] **Ge Yan** [1] **Jiaming Shan** [1] **Wenjie Wu** [1] **Junchi Yan** [1]

Abstract

Automatic quantum architecture search (QAS) has been widely studied across disciplines with different implications. In this paper, beyond a particular domain, we formulate the QAS problem into two basic (and relatively even ideal) tasks: i) arbitrary quantum circuit (QC) regeneration given a target QC; ii) approximating an arbitrary unitary (oracle). The latter can be connected to the setting of various quantum machine learning tasks and other QAS applications. Based on these two tasks, we generate a public QAS benchmark including 900 random QCs and 400 random unitary matrices which is still missing in the literature. We evaluate six baseline algorithms including brute force search, simulated annealing, genetic algorithm, reinforcement learning, hybrid algorithm, and differentiable algorithm as part of our benchmark. One characteristic of our proposed evaluation protocol on the basic tasks is that it deprives the domain-specific designs and techniques as used in existing QAS literature, making a unified evaluation possible and focusing on the vanilla search methods themselves without coupling with domain prior. In fact, the unitary approximation task could be algorithmically more difficult than the specific problems as it needs to explore the whole matrix space to fit the unitary. While specific tasks often only need to fit a partial observation of the unitary as the objective for search. Data and code are available at `https://github.com/Lucky-Lance/QAS-Bench`.

1. Introduction

Quantum computing has shown its promising potential in solving complex problems which are intractable for classical computers, with successful quantum algorithms such as the Grover algorithm (Grover, 1996) for database search and Shor algorithm (Shor, 1994) for prime factorization.

Especially with the arrival of the so-called Noisy Intermediate-Scale Quantum (NISQ) era (Preskill, 2018), more algorithms (Arute et al., 2019; Wu et al., 2018) are proposed to explore quantum supremacy (Preskill, 2012) by landing on a NISQ device. Designing a hardware-efficient quantum circuit (QC) for a certain algorithm requires substantial human effort and takes a long time. Thus, lots of quantum algorithms leave the oracle directly on the QCs, which is a black box with the ability to yield certain solutions but the internal circuit structure is not clearly stated. Therefore, a series of works have focused on how to automatically design a QC (Duong et al., 2022; He et al., 2022) to ease the burden of manually designing QCs.

In this paper, we focus on the problem of automatically designing QCs, namely QAS. The essence of searching for a QC is to decide how to arrange different quantum gates in the circuit. A QC is used to evolve a quantum system, which can be described by a unitary matrix. Thus, QAS can be mathematically defined as a unitary approximation problem. The approximation of an arbitrary unitary is generally difficult (Nielsen & Chuang, 2002), since we need exponentially many operations to generate an arbitrary state of n qubits. Therefore, the unitary approximation is non-polynomial in computational complexity, indicating that there are no polynomial QAS algorithms.

Moreover, QAS has wide applications in quantum computing. Firstly, it can automatically design circuits, such as quantum adders (Li et al., 2017) or quantum oracles in quantum algorithms such as those in Grover algorithm and Quantum Fourier Transform (QFT) (Zhang et al., 2022). Secondly, it can tackle quantum error correction in quantum information (Rigby, 2021), which has been considered the most important tool to exert quantum supremacy. Thirdly, it is also capable of searching for VQE ansatze, which requires optimizing both the architecture and the rotation parameters at the same time. MaxCut in Combinatorial Optimization (CO) (Guerreschi & Matsuura, 2019) and ground state energy estimation in quantum many-body problems (Barkoutsos et al., 2018) has received lots of attention. Last but not least, QAS can search for the Parameterized Quantum Circuit (PQC) in numerous quantum neural networks (QNN)

[1]MoE Key Lab of AI, Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Junchi Yan <yanjunchi@sjtu.edu.cn>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

(e.g. QCNN for image classification tasks) (Cong et al., 2019). Different from VQE which evolves the quantum system for the lowest eigenvalue, QNN is designed analog to a classical neural network, resulting in more complicated settings, such as supervised and unsupervised learning.

There are now a number of works available for each of the applications of QAS. These works are used to measure the performance of the algorithms on one or several tasks (e.g. image classification, CO problems, etc). There still lacks a benchmark for measuring the ability of these algorithms across tasks, or under the general QAS task in a broad sense.

Even for a particular task, how to evaluate the performance of a quantum search algorithm is not unified. Taking ground state energy estimation as an example, we need to design a QC given a molecule Hamiltonian and then evolve the quantum state to the eigenstate with minimum eigenvalue, which is the ground state energy of the molecule. The result is considered admissible if the energy is within the chemical accuracy (1 kcal/mol) (Klimeš et al., 2009). Whether achieving chemical accuracy is enough and then searching for a shallower circuit or we should try to estimate the ground state energy as close to the ground truth as possible still remains a problem (Ostaszewski et al., 2021). (Du et al., 2022; Wang et al., 2022b) test their search algorithms under circuit noise, which results in energy much higher than the chemical accuracy. Different metrics and perspectives of the problem lead to strong difficulty in identifying which searching approach is better even for this particular ground state energy estimation task.

Current QAS algorithms are often designed to solve related cross-disciplinary problems through quantum computing. For example, estimating the ground state energy of molecules requires knowledge of quantum chemistry (Levine et al., 2009), and solving CO problems (e.g. MaxCut) with VQE requires knowledge of graph theory (West et al., 2001). As illustrated in (Schatzki et al., 2021), when benchmarking quantum machine learning methods, it is better to provide data in quantum form which requires no embedding scheme, other than classical data (e.g. MNIST, MaxCut). For quantum architecture search problems, it is desirable that we are able to abstract the most essential features from these cross-disciplinary problems, thus getting rid of the need for other expertise. Thus, we decide to use the unitary approximation problem to evaluate the QAS algorithms. The proposed benchmark is to examine the ability of different algorithms to search for a QC given the unitary.

It is well worth noticing that the above-mentioned QAS is similar to the concept of Neural Architecture Search (NAS), which is a popular and important research field in machine learning. NAS is a technique for automatically designing neural networks and there have been lots of recent works on how to design efficient search strategies for a high-performance neural network (Liu et al., 2018b;a; Jaafra et al., 2019; Liu et al., 2022). Having a good knowledge of these approaches will be helpful for solving QAS problems.

In this paper, we propose a benchmark for QAS called QAS-Bench, with two protocols to evaluate the effectiveness of QAS: QC Regeneration from an arbitrary QC, and Unitary Approximation from an arbitrary unitary matrix. Accordingly, a dataset containing 900 QCs and 400 quantum unitary matrices is provided. For the proposed QAS-Bench, there is no open-source code that can be directly used, so we re-implemented baseline algorithms in (Williams & Gray, 1999; Ostaszewski et al., 2021; Du et al., 2022; Zhang et al., 2022) into PyTorch, forming a benchmark with traditional search algorithms as well as emerging learning-based methods. The dataset as well as the source code will be made publicly available. To sum up, the main contributions of our work are:

1) We revisit the quantum architecture search (QAS) problem as widely studied in literature with different implications, and conclude with two general tasks for QAS: QC Regeneration and Unitary Approximation whereby the searched QC can either be parameterized or not.

2) For the above two tasks, by random generation with physically meaningful post-processing, we manage to provide a benchmark dataset containing 900 QCs for QC Regeneration and 400 unitary matrices for Unitary Approximation, respectively. The evaluation protocols are also defined. Perhaps more importantly our released source code can be used by third-party to generate more diverse and challenging benchmarks beyond the scope of our current version.

3) We further evaluate six representative baselines on our benchmark. To our best knowledge, QAS-Bench is the first public benchmark for QAS, which may well facilitate future research in this field.

Remark on the QAS problem for QML:

To the broad machine learning community, readers may show curiosity or even concern for the relation of the task of QML (e.g. QNN for a tailored training dataset) to our QAS-Bench.

Firstly, QAS is capable of searching QNN structures due to its ability of designing PQCs. The original paper of the hybrid algorithm (Du et al., 2022) (re-implemented in our benchmark) searches a QNN for image classification tasks. For our re-implemented baseline algorithms, they can search QNNs by replacing the loss function related to the given unitary with the one concerning training set labels.

Secondly, in the Unitary Approximation task we form a train set by sampling input-output quantum state pairs, which can be used for the search of QNNs.

2. Quantum Architecture Search Background

QAS has been widely studied across disciplines and has been referred in different settings with different implications and names (i.e. QC decomposition (Bharti et al., 2022), adaptive circuit (Grimsley et al., 2019), quantum ansatz search (Zhang et al., 2022), etc.). Therefore, we review all the QAS-related definitions as well as their applications in this section. The essence of all these works is to automatically design QCs for a certain problem. We can divide these applications by whether the problem provides a unitary matrix. Searching with a given unitary matrix is more difficult since the unitary matrix is the strongest restriction compared to the input-output pairs or minimizing a certain objective function under the given observation.

2.1. Applications with Unitary Matrices

Quantum oracle is a black box whose internal circuit structures are not clearly stated, but with the ability to yield solutions to a certain problem. Quantum oracle is of fundamental significance in quantum computing and quantum information, but implementing an arbitrary quantum oracle requires exponentially many gates (Nielsen & Chuang, 2002). Most of the quantum oracles are designed on a circuit without parameterized gates but we can also use PQCs to implement the oracles. Researchers have put continuous efforts into manually designing quantum oracles (Bijwe et al., 2022; Rahman & Paul, 2022), but the emergence of QAS provides a more effective paradigm for quantum oracle implementation. The early work (Ding et al., 2006) tries to evolve quantum oracles by search algorithms. (Li et al., 2017; Deibuk & Biloshytskyi, 2015) design quantum adders, as further leveraged for the construction of quantum autoencoders in (Lamata et al., 2018). Recently the QFT oracle is designed in (Zhang et al., 2022).

2.2. Applications without Unitary Matrices

Quantum error correction (QEC) is widely accepted as the key to fault-tolerant quantum computation, as it can protect quantum information from errors due to environmental noise and experimental imperfections (Cai et al., 2021). Quantum error correction code is one of the QEC approaches which can be used to encode the quantum states of qubits in case of potential errors during communication over quantum channels. Typical QEC code includes bit flip code (Peres, 1985) and Shor code (Shor, 1995), etc. With the ability to automatically design circuits, QAS can efficiently design QEC codes. There have been recent works for automatically designing QEC codes and their decoders, such as heuristic search methods (Rigby, 2021), machine learning methods (Chen et al., 2019; Cong et al., 2019) as well as reinforcement learning (RL) methods (Nautrup et al., 2019; Zeng et al., 2022).

Variational quantum eigensolver (VQE) (Peruzzo et al., 2014) is a quantum algorithm which has been widely used for quantum chemistry, optimization problems, and quantum simulation. It trains a PQC using a classical optimizer to solve matrix eigenvalues and eigenvectors. The automatic design of a VQE circuit involves searching both architecture parameters (find a suitable gate arrangement for a PQC) and rotation parameters (set the angles of rotation gates on the PQC). QAS can search for a VQE circuit by optimizing architecture and rotation parameters iteratively. For ground state energy estimation in quantum chemistry, Unitary Coupled-Cluster (UCC) theory uses Trotter-Suzuki decomposition to obtain the QC for a given molecule Hamiltonian (Barkoutsos et al., 2018), which usually leads to large circuits. There have been works on automatically improving UCC ansatze (Grimsley et al., 2019; Sapova & Fedorov, 2022), or automatically searching for PQCs without given ansatze (Ostaszewski et al., 2021; Wang et al., 2022b). For MaxCut in CO problems, quantum approximate optimization algorithm (QAOA) (Farhi et al., 2014) proposes an ansatz using Ising model, and various methods have been proposed to further optimize the QAOA ansatz for MaxCut (Majumdar et al., 2021a;b). Recently, (Duong et al., 2022; Zhang et al., 2022) have tried to introduce QAS to automatically search for VQE ansatze for MaxCut.

Quantum neural network (QNN) is a type of artificial neural network that can be implemented by PQCs. Quantum convolution neural networks (Cong et al., 2019) and recurrent quantum neural networks (Bausch, 2020) have been proposed in recent years, achieving comparable results with classical neural networks. As opposed to the classical neural network, QNN has faster training speed and higher predicting accuracy (Huang et al., 2022). QNNs can handle tasks such as image classification (Mathur et al., 2021; Wang et al., 2021). There are some recently proposed automatic QNN design methods (Zhang et al., 2021; Wang et al., 2022b; Duong et al., 2022), expecting to improve flexibility and accuracy by reducing human intervention.

2.3. QAS Algorithms

Here we briefly classify the current QAS baselines into four main categories: evolutionary algorithms, differentiable search, reinforcement learning (RL) methods, and hybrid methods. We leave the detailed description in Appendix A.

3. Proposed Datasets and Metrics

We propose two datasets, namely QC Regeneration, and Unitary Approximation, with evaluation protocols respectively. Note that approximating an arbitrary unitary is more difficult than regenerating an arbitrary QC, since regenerating an arbitrary QC only requires a bounded number of quantum gates and we guarantee that there exists a valid cir-

cuit structure. But for an arbitrary unitary matrix U with n qubits, (Nielsen & Chuang, 2002) provides a method to approximate it within a distance ε by $O(n^2 4^n \log^c(n^2 4^n/\varepsilon))$ gates, and proves that the lower bound of needed gate number is $\Omega(\frac{2^n \log(1/\varepsilon)}{\log(n)})$. To test the search ability of different QAS algorithms in real physical scenarios, we have now also opened a new dataset in which we add unitary matrices with practical meaning to the dataset (such as Bell state and GHZ state). Users of QAS-Bench can further add more unitary matrices into our dataset according to our instructions.

3.1. QC Regeneration

We provide randomly generated circuits as well as their corresponding unitary matrices. An algorithm is required to generate a circuit which is equivalent to the original circuit given its unitary under the same candidate gate set. We split this dataset into two folds, RandomCircuit-Single (RC-S) and RandomCircuit-Clifford (RC-C), with different candidate gate sets. To be more specific, RC-S is generated with candidate gate set $\mathcal{G}_S = \{H, S, T, I\}$ and RC-C with $\mathcal{G}_C = \{H, S, T, I, \text{CNOT}\}$.

$\mathcal{G}_C$ is a commonly used universal gate set (Clifford) which can be used to approximate any unitary (Nielsen & Chuang, 2002). $\mathcal{G}_S$ removes the CNOT gates from $\mathcal{G}_C$ and forms a relatively easier setting. We can further add new tasks to the QC Regeneration dataset with other candidate gate sets.

3.1.1. Dataset Generation

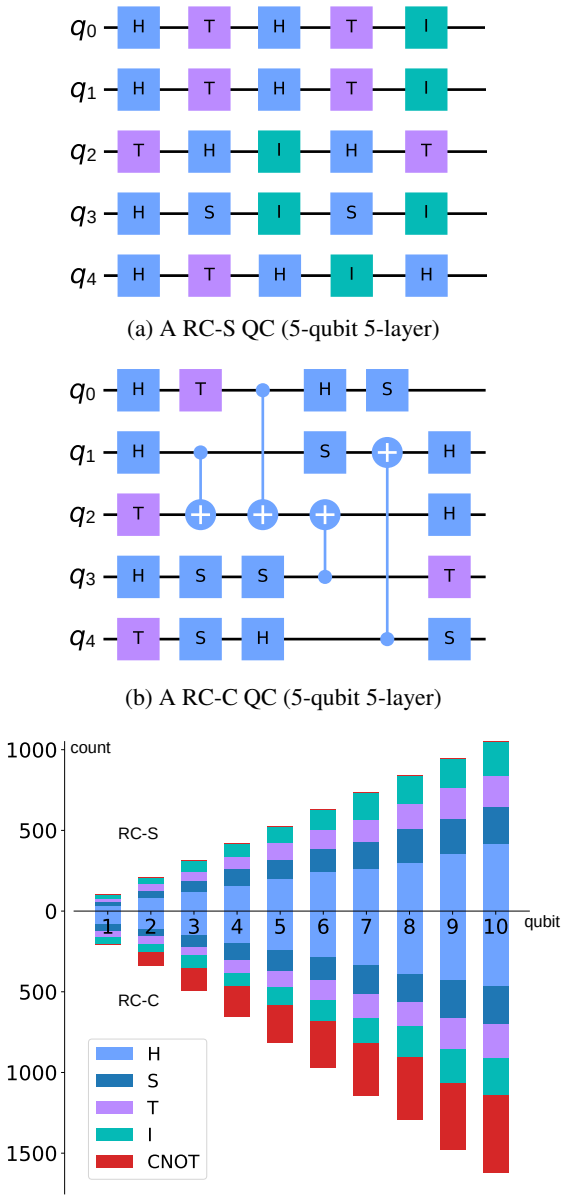
We illustrate in detail how to randomly generate a QC by a given candidate gate set $\mathcal{G}$. Consider an n-qubit m-layer circuit, assume there are $N(j)$ gates in layer j, where $N(j) \leq n$, we denote the i-th gate in the j-th layer as $\boldsymbol{M}_{ij}$. For the sake of clarity, we define $\boldsymbol{U}_{ij} = \sigma(\boldsymbol{M}_{ij})$ which maps a quantum gate to a $2^n \times 2^n$ unitary with all the irrelevant qubits kept as identity gates (I gates). The unitary of this generated circuit is

$$\boldsymbol{U}_t = \prod_{j=1}^{m} \prod_{i=1}^{N(j)} \boldsymbol{U}_{ij} = \prod_{j=1}^{m} \prod_{i=1}^{N(j)} \sigma(\boldsymbol{M}_{ij}) \qquad (1)$$

When generating a circuit, the qubit number and layer number should be specified first. Then given the candidate gate set $\mathcal{G}$, we need to give each gate $g \in \mathcal{G}$ a weight proportional to its probability of occurrence in the circuit. Based on the probability of each gate we randomly generate a QC.

After generating a circuit, we need to avoid circuit redundancy, and thus, we apply a simple optimization for the generated circuit. We mainly check for sequential gates resulting in a new gate which is also in $\mathcal{G}$ (e.g. two CNOT gates or two H gates are equivalent to one I gate, and two T gates are equivalent to one S gate). For redundancy with one-qubit gates, we maintain the first gate and regenerate a new one-qubit gate to replace the second one. For those with two-qubit gates (i.e. CNOT gate), we flip the control and target bit of the second gate. Our optimization does not necessarily guarantee the circuit to be optimal, but it can reduce circuit redundancy to a large extent. Then the unitary of the generated circuit can be calculated by Eq. 1. Examples of our generated circuits are shown in Fig. 1

In the final dataset for QC re-generation, we provide 60 subtasks by generating QCs with qubits from 1 to 10 with layers from 1 to 6. In each subtask, we provide 5 RC-S



(a) A RC-S QC (5-qubit 5-layer)

(b) A RC-C QC (5-qubit 5-layer)

(c) Distribution of each type of gates

Figure 1: Overview of dataset QC Regeneration

circuits and 10 RC-C circuits. The total number of circuits sums up to 900. We give visualization of the generated circuits by Qiskit (Aleksandrowicz et al., 2019), and the corresponding unitary of each circuit is also provided.

3.1.2. Evaluating Metric

We present a protocol to judge whether two circuits are equivalent. It involves a distance L to represent the difference between the target matrix $\boldsymbol{U}_t$ and matrix $\boldsymbol{U}_s$ for the searched circuit. The distance L is the same as the fitness function provided in (Williams & Gray, 1999). For a matrix of n qubits, L is calculated by

$$L = \sum_{i=0}^{2^n-1} \sum_{j=0}^{2^n-1} |\boldsymbol{U}_{s_{ij}} - \boldsymbol{U}_{t_{ij}}|. \tag{2}$$

where $\boldsymbol{U}_s \in \mathbb{C}^{2^n \times 2^n}$, $\boldsymbol{U}_t \in \mathbb{C}^{2^n \times 2^n}$ and $|\cdot|$ is the modulus of a complex number.

If the distance L of two matrices is less than a given threshold ϵ, we consider these two matrices are identical[1].

3.2. Unitary Approximation

As stated in Sec. 2, there are various QAS applications where the unitary matrices are not necessarily available (e.g. QNN). Thus, randomly generated unitary matrices as well as the sampled training sets are included in the dataset to fit such cases. The sampling details are given in Sec. 3.2.1. A new evaluation metric based on the test set is also provided. In this dataset, an algorithm is required to search for a QC which conforms to the given metric. We provide two searching schemes, one can directly search for a QC given unitary, or train the algorithm based on the train set.

3.2.1. Dataset Generation

All of the unitary matrices are generated arbitrarily by the Python package SciPy (Virtanen et al., 2020) which uses the algorithm proposed in (Mezzadri, 2006) to randomly generate matrices. To be more specific, the real part and imaginary part of matrices are generated separately. The two matrices are summed up, and then convert to a unitary by applying the QR decomposition (Gander, 1980). Without loss of generosity, to ease the difficulty of the Unitary Approximation dataset such that it can be effectively searched by commonly used candidate gate sets such as $\mathcal{G}_R = \{R_x(\theta), R_y(\theta), R_z(\theta), \mathrm{CNOT}\}$ in (Du et al., 2022; Ostaszewski et al., 2021), we set the determinants of the matrices to 1. (proved in Lemma. C.1 and Lemma. C.2).

From 2 to 5 qubits, we generate 100 unitary matrices for each number of qubits, and the total number of matrices sum up to 400 in this dataset. The corresponding train set $\mathcal{D}$ and

[1]We set $\epsilon = 10^{-10}$ considering the precision limit in Python.

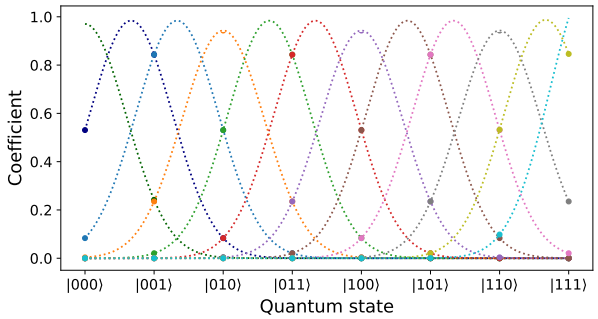


Figure 2: Coefficient w.r.t Quantum state. The dots in each curve represent the coefficient number of each state $|j\rangle$ in Eq.5, and each curve corresponds to i (i means the index of the sampled state $|\psi\rangle$). The peak is not necessarily at 1 since we normalize the coefficients.

the test set $\mathcal{V}$ contain input and output quantum state pairs. We randomly sample the input states based on the following three different sampling schemes. We first illustrate in detail how to generate $\mathcal{V}$.

Firstly, we add a set of computational basis states, where

$$|\psi_i\rangle = |i\rangle\,, \quad i = 0, 1, \cdots, 2^n - 1 \tag{3}$$

Secondly, we sample from other pure states, and the coefficient numbers of the state vector are sampled from a Gaussian distribution. The center of the Gaussian distribution transfers smoothly from state $|00\cdots0\rangle$ to state $|11\cdots1\rangle$. The size of the Gaussian part N_{G} is set to

$$N_{\mathrm{G}} = \frac{|\mathcal{V}| - 2^n}{2} \tag{4}$$

Thus, each input state can be denoted as

$$|\psi_i\rangle = \sum_{j=0}^{2^n-1} \frac{\mathcal{N}(\mu_i, \sigma^2)(j)}{\sqrt{\sum_{k=0}^{2^n-1} \mathcal{N}^2(\mu_i, \sigma^2)(k)}} |j\rangle \tag{5}$$

where $\mu_i = \frac{i}{N_{\mathrm{G}}} \cdot 2^n$, $\sigma = 0.6$, and $i = 0, 1, \cdots, N_G - 1$. The process of sampling the coefficient numbers in a 3-qubit situation is shown in Fig. 2.

Thirdly, we add imaginary numbers to the coefficient numbers of the input state vector. The size of this part of test sets N_{C} is the same as N_{G}. Each coefficient number is a random complex number generated by Euler's formula

$$C_{i,j} = \sqrt{\alpha} \cdot \exp(\beta \mathrm{i}) \tag{6}$$

where i is the imaginary unit, $\alpha \sim U(0,1), \beta \sim U(0, 2\pi)$. $U(\cdot,\cdot)$ means the uniform distribution. We first sample all the complex numbers from $C_{i,0}$ to $C_{i,2^n-1}$, then normalize

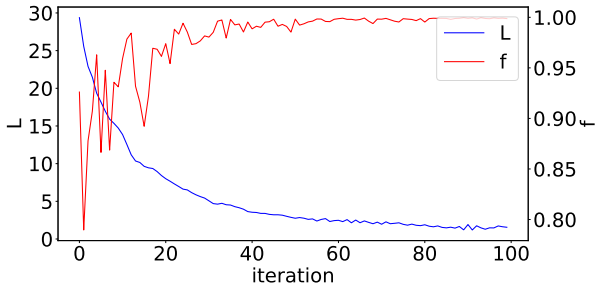


Figure 3: L w.r.t f. As we can see, distance L and f show a symmetric pattern and reach the respective extremums after certain iterations.

them. The state vector of each input state is

$$|\psi_i\rangle = \sum_{j=0}^{2^n-1} \frac{C_{i,j}}{\sqrt{\sum_{k=0}^{2^n-1} |C_{i,k}|^2}} |j\rangle \tag{7}$$

where $i = 0, 1, \cdots, N_C - 1$. The output state is obtained by multiplying the input state with the provided unitary matrix

$$|\psi_{\text{out}}\rangle = \boldsymbol{U}_t |\psi_{\text{in}}\rangle \tag{8}$$

For the different number of qubits, the size of the test set $\mathcal{V}$ varies. We provide 32 pairs of quantum states for 2-4 qubits each and 64 pairs for 5 qubits.

As for the train set $\mathcal{D}$, we dismiss the basis states (Eq. 3), and the sampling process of the input quantum states follows Eq. 5 and Eq. 7. We set original $N_G = N_C = 100$ for 2-4 qubits, $N_G = N_C = 200$ for 5 qubits and then sample input quantum states. We remove the duplicated states that appear both in the train set and in the test set by replacing them with randomly generated states by Eq. 7. The corresponding output states can be calculated by Eq. 8.

All the unitary matrices and the corresponding $\mathcal{D}$ and $\mathcal{V}$ are provided in this Unitary Approximation dataset. We will add unitary matrices together with train and test sets for more qubits in the future.

3.2.2. Evaluating Metrics

Compared to the QC Regeneration dataset, it is much harder for an algorithm to achieve a small ϵ distance with L (Eq. 2) in the Unitary Approximation dataset. Moreover, L increases exponentially with the qubit number n, which leads to a huge distance and no evaluation criteria. Therefore we define a metric with physical meaning in $\mathcal{V}$ to estimate the approximation ability.

We define a new metric f to evaluate the similarity between the predicted state $|\phi\rangle$ and ground-truth state $|\psi\rangle$. For a predicted state $|\phi\rangle = \sum_{j=0}^{2^n-1} (\hat{a}_j + \hat{b}_j \mathbf{i}) |j\rangle$ and ground-truth state $|\psi\rangle = \sum_{j=0}^{2^n-1} (a_j + b_j \mathbf{i}) |j\rangle$, f can be calculated by the following equation:

$$f = \left(\sum_{j=0}^{2^n-1} \sqrt{a_j^2 + b_j^2} \cdot \sqrt{\hat{a}_j^2 + \hat{b}_j^2} \right)^2 \tag{9}$$

If $f = 1$, then we have an equal possibility of getting the same basis state when measuring the predicted and ground-truth state. The relationship between L and f in one searching process is shown in Fig. 3.

4. Experiment and Results

All experiments were carried out on a workstation with Intel(R) Xeon(R) Platinum 8276 CPU and 4 NVIDIA A100-PCIE-40G GPUs. The experiments are conducted under a full amplitude quantum simulator without circuit noise, and all codes are written in Python. We implement baseline algorithms based on a revised version of simulator TorchQuantum (Wang et al., 2022a) with a better support for PyTorch.

4.1. Baselines for Evaluation

We select and implement six baselines and provide them along with the benchmark datasets:

1) Brute Force Search (BFS) is the most basic algorithm used for the comparison among all baselines. Since brute force search contains only enumeration and evaluation, it is capable of reflecting the complexity of the dataset. We design a naive BFS for both two datasets and add a bidirectional version for QC Regeneration dataset.

2) Simulated Annealing (SA) is a commonly used heuristic algorithm which simulates the process of minimizing the energy of a material by heating then slowly lowering the temperature. We adopt the simulated annealing algorithm to compare different heuristic methods.

3) Genetic Algorithm (GA) is another heuristic algorithm which views the search as the process of natural selection. It uses generation updating methods to update searching results iteratively. (Williams & Gray, 1999) uses genetic algorithm to design QCs, we implement the genetic algorithm based on this method.

4) Reinforcement learning (RL) in (Ostaszewski et al., 2021) encodes quantum gate types and gate position into tuples which represent valid actions at each step, using a feedback-driven curriculum learning method to estimate the ground state energy of LiH. We re-implement the code into our QAS setting.

5) Hybrid Algorithm (HA) proposed in (Du et al., 2022) automatically designs QNNs. It samples several ansatze

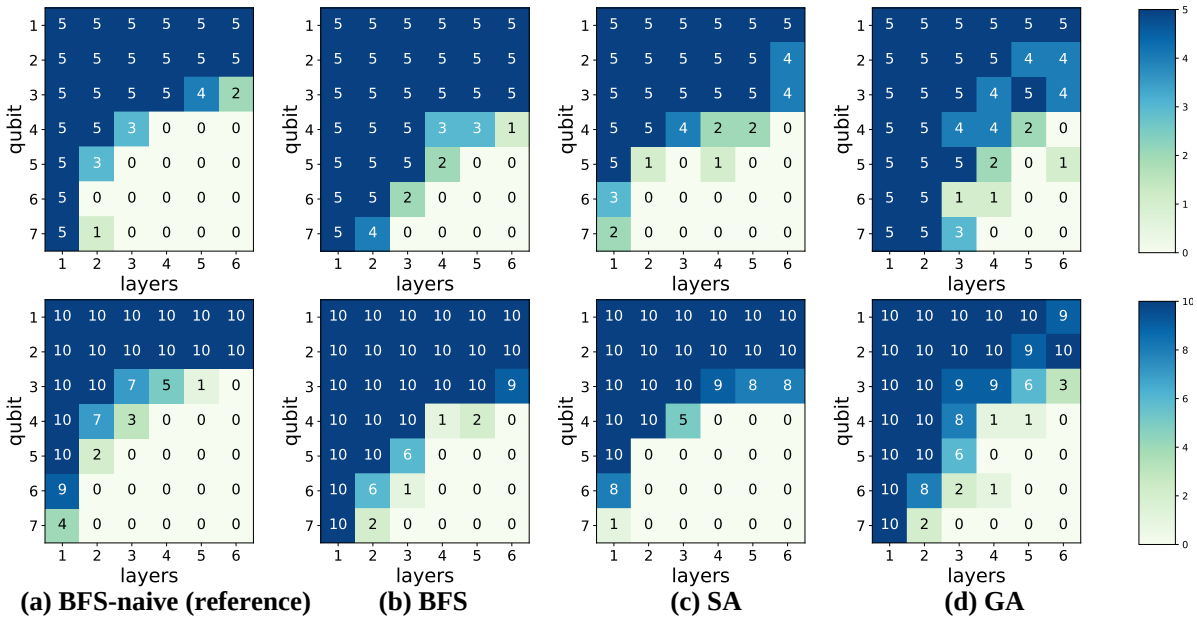


Figure 4: QC Regeneration results of traditional searching algorithms. **Top:** results of 3 baselines on the RC-S dataset which has 5 circuits for each layer (1-6) of each qubit (1-7); **Bottom:** results of baselines on the RC-C data which has 10 circuits for each layer of each qubit (BFS-naive as a reference).

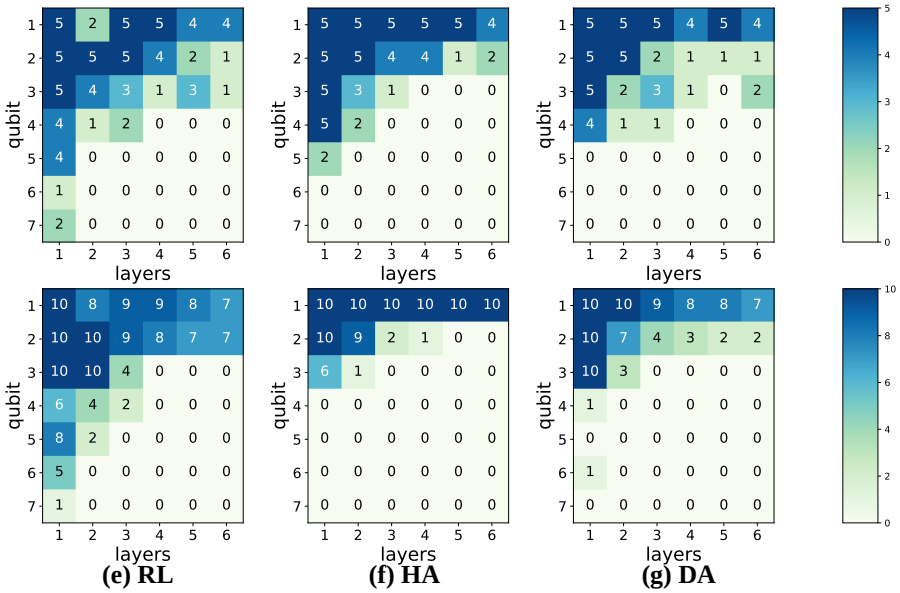


Figure 5: QC Regeneration results of machine learning related algorithms. The arrangement of figure is the same as Fig. 4.

(architecture parameters), optimizes the rotation parameters in each ansatz, and ranks the searching results. The circuit ranking first is then finetuned and selected as the output. We re-implement it into PyTorch and make some revisions to the code to fit our QAS setting.

6) Differentiable Algorithm (DA) in (Zhang et al., 2022) transfers the sampling process into a differentiable part and assembles it into the neural network. It designs a Monte Carlo gradient for structure sampling and calculates it by a mean-field probability model. Then the architecture parameters and rotation parameters can be updated by gradient propagation iteratively. We change the code into PyTorch and revise it to fit our QAS setting.

Further information and implementation details about the

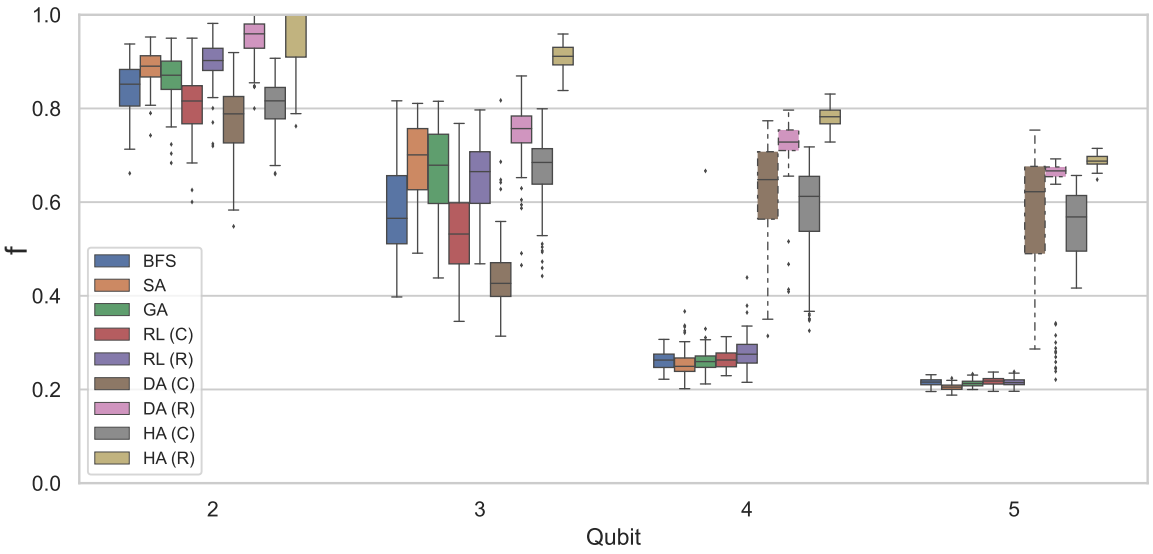


Figure 6: Result of Unitary Approximation. In the legend of the figure, (C) refers to the candidate gate set $\mathcal{G}_C$, and (R) refers to the candidate gate set $\mathcal{G}_R$. The dashed box of DA with 4 and 5-qubit means we omit the designed Monte Carlo gradient and just use Softmax over architecture parameters as discussed in Sec. B.5.

baseline algorithms will be introduced in Appendix B.

4.2. QC Regeneration

We test the above-mentioned baselines on both the RC-S and RC-C datasets, and the results are illustrated in Fig. 4 and 5. Following the metric in Sec. 3.1.2, we set the candidate gate set for RC-S as $\mathcal{G}_S = \{H, I, S, T\}$ and RC-C as $\mathcal{G}_C = \{H, I, S, T, \text{CNOT}\}$. Due to page limit of the paper, we only report the results up to 7 qubits.

From Fig. 4, we can see that the BFS-naive algorithm has set a standard about the edge of the searching space against the computation power of our environment. We find it takes more than 48 hours to iterate through all the possibilities for a only 7-qubit 2-layer circuit. Notice that the proposed bidirectional BFS in the baseline can have a squared speedup than the naive breadth-first search algorithm, which leads to a sharp increase in algorithm performance.

Two heuristic algorithms both outperform BFS-naive and achieve reasonable results, showing similar performance compared to the bidirectional BFS. This result proves why genetic algorithm has been studied for QAS problems since 1999 and is still widely used as a robust solution to QAS. We improve the chromosome selection and generation management of the original genetic algorithm. The results from the simulated annealing suggest that heuristic algorithms can always be considered as reliable solutions and the genetic algorithm is more suitable to the QAS problem.

RL shows a completely different pattern than the previous ones. The results demonstrate that RL has a greater randomness, which might be related to the searching scheme of RL. Moreover, the performance of RL algorithms is closely related to the reward system. We believe RL has the potential to perform better with a better reward.

The hybrid algorithm surprisingly fails on this regeneration dataset. The hybrid algorithm focuses on two steps optimization for PQCs and the SOTA hybrid algorithm we use here is sampling based. With no parameterized rotation gates in the candidate gate set, the hybrid algorithm we use degenerates to a simple sampling method, which leads to poor performance. A similar circumstance is with the differentiable algorithm which simply designs a gradient for sampling. The differentiable search algorithm achieves similar performance with hybrid algorithm. The results also suggest that the circuit sampling part of the proposed baselines is not working well without further tuning the rotation parameters, which indicates the architecture search part of the baseline can be further improved.

4.3. Unitary Approximation

We further test our baselines on the proposed Unitary Approximation dataset, which is much harder than the QC Regeneration dataset. We make direct use of the unitary for search and test set for evaluation, and provide two independent experimental settings with two candidate gate sets. The first one uses the Clifford group $\mathcal{G}_C = \{H, I, S, T, \text{CNOT}\}$ to approximate the unitary matrices. All six baselines are tested under this experimental setting. The second one uses $\mathcal{G}_R = \{R_x(\theta), R_y(\theta), R_z(\theta), \text{CNOT}\}$ with parameterized rotation gates. Since BFS and two heuristic algorithms are

not capable of optimizing both architecture parameters and rotation parameters together, we only test the RL, hybrid algorithm and differentiable algorithm under this setting. Thus, we have nine baseline results reported in Fig. 6.

The searching process for the Unitary Approximation dataset differs from QC Regeneration as there is no information of layer number provided. For brute force search, we search from 1 gate, then iteratively add the gate number until the circuit is found or the time limit is exceeded. For other algorithms, we provide two ways of setting the layer number. The first one is fixing a layer number for each algorithm. Another is to set a layer number that grows linearly with the number of qubits for each algorithm. In our setting, for a n-qubit circuit, we set the initial layer number to $5n$.

From Fig. 6 we can see that hybrid algorithm outperforms all other baselines and achieves consistent results. BFS still sets a standard for all baselines but the performance is affected by the limitation of circuit depth. As the number of qubits increases, BFS can only iterate through all the possibilities for less deep circuits. Thus, the performance of BFS has a substantial decline and f is around 0.2 for 5-qubit unitary matrices. Simulated annealing and genetic algorithm show the same pattern as the BFS.

The slump continues for the RL from QC Regeneration to Unitary Approximation. The results of RL are slightly better than the BFS and still not comparable to the other 2 machine learning algorithms. Considering the performance on ground state energy estimation, we still believe the RL is a promising QAS approach but it overly depends on the design of the reward system. However, the results exhibit that using $\mathcal{G}_R$ can slightly improve performance.

The hybrid algorithm and differentiable algorithm have a huge rebound and demonstrate the power of updating both architecture and rotation parameters. These two algorithms can still achieve a result more than 0.6 with 5-qubit unitary matrices. This explains why hybrid/differentiable algorithms, especially those imitating the DARTS (Liu et al., 2018b) framework, have achieved increasing attention. Sec. B.4 mentions that the hybrid algorithm samples from predefined ansatz structures, so it can exceed the differentiable algorithms (with no prior knowledge) to some extent. Equipped with the ability to adjust rotation angles, DA (R) outperforms DA (C) from 2-qubit tasks to 5-qubit tasks, and HA (R) also outperforms HA (C), this shows the importance of updating rotation parameters after sampling structures. The result also implicates that the design of the gradient for structural parameters is not yet very effective (no better than simple sampling in our benchmark). From this we conclude that architecture parameters that work better may be more important than the rotation parameters, and how to effectively search for the architecture parameters of quantum circuits can be left as future work.

4.4. Results with Simulated Circuit Noise

TorchQuantum currently does not have complete support for noise models, and we find it hard to take all common noise models into consideration. For simplicity we add a readout noise and details are showed in Appendix D. As expected, all algorithms have a reduced performance after adding circuit noise. We call for researchers to add more noise models and improve our benchmark together.

4.5. Discussion

Based on the results we can conclude that: **1)** Classical search algorithms such as brute force search, heuristic algorithms and genetic algorithms are by far competitive and robust baselines. **2)** In the QAS algorithms, the rotation parameters are quite important and can substantially improve the search ability. **3)** The large performance difference of hybrid algorithm and differentiable algorithm on two datasets indicates that the sampling part of QAS algorithms may well limit their performance. How to break the boundary between discrete sampling and continuous optimization is an important problem in quantum computation.

5. Conclusion

We introduce the QAS-Bench benchmark for Quantum Architecture Search (QAS), a promising yet challenging way of automatically generating QCs which already has wide applications in quantum computing. We summarize the applications of QAS, classify the commonly used QAS methods, abstract the unitary approximation essence of QAS and formulate it into two basic tasks: QC Regeneration given arbitrary circuit and Unitary Approximation given arbitrary unitary. To be specific, QAS-Bench provides 2 datasets (containing 900 QCs and 400 unitary matrices) with evaluation protocols respectively. we also design/re-implement 6 baseline algorithms and analyze their results.

There are still some limitations to our work. We have temporarily not fully considered realistic instances and noise models in our benchmark. The most popular metric to evaluate the similarity between quantum states is quantum state fidelity calculated as $|\langle\psi|\phi\rangle|^2$ for state $|\psi\rangle$ and $|\phi\rangle$. However, with the qubit number grows, we find the results of all algorithms degrade to nearly 0. Thus we define f with physical meaning to better distinguish these baselines. In future work, we will design more efficient algorithms and include state fidelity as an evaluation metric for better scalability.

Acknowledgments

The work was supported in part by National Key Research and Development Program of China (2020AAA0107600) and NSFC (62222607).

References

Aleksandrowicz, G., Alexander, T., Barkoutsos, P., Bello, L., Ben-Haim, Y., Bucher, D., Cabrera-Hernández, F. J., Carballo-Franquis, J., Chen, A., Chen, C.-F., et al. Qiskit: An open-source framework for quantum computing. *Accessed on: Mar*, 16, 2019.

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G., Buell, D. A., et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

Barkoutsos, P. K., Gonthier, J. F., Sokolov, I., Moll, N., Salis, G., Fuhrer, A., Ganzhorn, M., Egger, D. J., Troyer, M., Mezzacapo, A., et al. Quantum algorithms for electronic structure calculations: Particle-hole hamiltonian and optimized wave-function expansions. *Physical Review A*, 98 (2):022322, 2018.

Bausch, J. Recurrent quantum neural networks. *Advances in neural information processing systems*, 33:1368–1379, 2020.

Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Alam, M. S., Ahmed, S., Arrazola, J. M., Blank, C., Delgado, A., Jahangiri, S., et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

Bharti, K., Cervera-Lierta, A., Kyaw, T. H., Haug, T., Alperin-Lea, S., Anand, A., Degroote, M., Heimonen, H., Kottmann, J. S., Menke, T., et al. Noisy intermediate-scale quantum algorithms. *Reviews of Modern Physics*, 94(1):015004, 2022.

Bijwe, S., Chauhan, A. K., and Sanadhya, S. K. Implementing grover oracle for lightweight block ciphers under depth constraints. In *Australasian Conference on Information Security and Privacy*, pp. 85–105. Springer, 2022.

Cai, W., Ma, Y., Wang, W., Zou, C.-L., and Sun, L. Bosonic quantum error correction codes in superconducting quantum circuits. *Fundamental Research*, 1(1):50–67, 2021.

Casale, F. P., Gordon, J., and Fusi, N. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116*, 2019.

Chen, H., Vasmer, M., Breuckmann, N. P., and Grant, E. Machine learning logical gates for quantum error correction. *arXiv preprint arXiv:1912.10063*, 2019.

Cong, I., Choi, S., and Lukin, M. D. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, 2019.

Deibuk, V. G. and Biloshytskyi, A. V. Design of a ternary reversible/quantum adder using genetic algorithm. *International Journal of Information Technology and Computer Science*, 7(9):38–45, 2015.

Ding, L. and Spector, L. Evolutionary quantum architecture search for parametrized quantum circuits. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 2190–2195, 2022.

Ding, S., Jin, Z., and Yang, Q. Evolving quantum oracles with hybrid quantum-inspired evolutionary algorithm. *arXiv preprint quant-ph/0610105*, 2006.

Dong, X. and Yang, Y. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1761–1770, 2019.

Du, Y., Huang, T., You, S., Hsieh, M.-H., and Tao, D. Quantum circuit architecture search for variational quantum algorithms. *npj Quantum Information*, 8(1):1–8, 2022.

Duong, T., Truong, S. T., Tam, M., Bach, B., Ryu, J.-Y., and Rhee, J.-K. K. Quantum neural architecture search with quantum circuits metric and bayesian optimization. *arXiv preprint arXiv:2206.14115*, 2022.

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

Gander, W. Algorithms for the qr decomposition. *Res. Rep*, 80(02):1251–1268, 1980.

Grimsley, H. R., Economou, S. E., Barnes, E., and Mayhall, N. J. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature communications*, 10(1):1–9, 2019.

Grover, L. K. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.

Guerreschi, G. G. and Matsuura, A. Y. Qaoa for max-cut requires hundreds of qubits for quantum speed-up. *Scientific reports*, 9(1):1–7, 2019.

He, Z., Chen, C., Li, L., Zheng, S., and Situ, H. Quantum architecture search with meta-learning. *Advanced Quantum Technologies*, 5(8):2100134, 2022.

Huang, H.-Y., Broughton, M., Cotler, J., Chen, S., Li, J., Mohseni, M., Neven, H., Babbush, R., Kueng, R., Preskill, J., et al. Quantum advantage in learning from experiments. *Science*, 376(6598):1182–1186, 2022.

Jaafra, Y., Laurent, J. L., Deruyver, A., and Naceur, M. S. Reinforcement learning for neural architecture search: A review. *Image and Vision Computing*, 89:57–66, 2019.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Klimeš, J., Bowler, D. R., and Michaelides, A. Chemical accuracy for the van der waals density functional. *Journal of Physics: Condensed Matter*, 22(2):022201, 2009.

Kuo, E.-J., Fang, Y.-L. L., and Chen, S. Y.-C. Quantum architecture search via deep reinforcement learning. *arXiv preprint arXiv:2104.07715*, 2021.

Lamata, L., Alvarez-Rodriguez, U., Martín-Guerrero, J. D., Sanz, M., and Solano, E. Quantum autoencoders via quantum adders with genetic algorithms. *Quantum Science and Technology*, 4(1):014007, 2018.

Levine, I. N., Busch, D. H., and Shull, H. *Quantum chemistry*, volume 6. Pearson Prentice Hall Upper Saddle River, NJ, 2009.

Li, R., Alvarez-Rodriguez, U., Lamata, L., and Solano, E. Approximate quantum adders with genetic algorithms: an ibm quantum experience. *Quantum Measurements and Quantum Metrology*, 4(1):1–7, 2017.

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., and Li, Z. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34, 2018a.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Liu, X., Zhao, J., Li, J., Cao, B., and Lv, Z. Federated neural architecture search for medical data security. *IEEE Transactions on Industrial Informatics*, 18(8):5628–5636, 2022.

Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G. G., and Tan, K. C. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*, 2021.

Majumdar, R., Bhoumik, D., Madan, D., Vinayagamurthy, D., Raghunathan, S., and Sur-Kolay, S. Depth optimized ansatz circuit in qaoa for max-cut. *arXiv preprint arXiv:2110.04637*, 2021a.

Majumdar, R., Madan, D., Bhoumik, D., Vinayagamurthy, D., Raghunathan, S., and Sur-Kolay, S. Optimizing ansatz design in qaoa for max-cut. *arXiv preprint arXiv:2106.02812*, 2021b.

Mathur, N., Landman, J., Li, Y. Y., Strahm, M., Kazdaghli, S., Prakash, A., and Kerenidis, I. Medical image classification via quantum neural networks. *arXiv preprint arXiv:2109.01831*, 2021.

Mezzadri, F. How to generate random matrices from the classical compact groups. *arXiv preprint math-ph/0609050*, 2006.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Nautrup, H. P., Delfosse, N., Dunjko, V., Briegel, H. J., and Friis, N. Optimizing quantum error correction codes with reinforcement learning. *Quantum*, 3:215, 2019.

Nielsen, M. A. and Chuang, I. Quantum computation and quantum information, 2002.

Ostaszewski, M., Trenkwalder, L. M., Masarczyk, W., Scerri, E., and Dunjko, V. Reinforcement learning for optimization of variational quantum circuit architectures. *Advances in Neural Information Processing Systems*, 34: 18182–18194, 2021.

Peres, A. Reversible logic and quantum computers. *Physical review A*, 32(6):3266, 1985.

Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.-H., Zhou, X.-Q., Love, P. J., Aspuru-Guzik, A., and O'brien, J. L. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):1–7, 2014.

Preskill, J. Quantum computing and the entanglement frontier. *arXiv preprint arXiv:1203.5813*, 2012.

Preskill, J. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

Prins, C. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

Rahman, M. and Paul, G. Grover on katan: Quantum resource estimation. *IEEE Transactions on Quantum Engineering*, 3:1–9, 2022.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pp. 2902–2911. PMLR, 2017.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Rigby, A. *Heuristics in quantum error correction*. PhD thesis, University of Tasmania, 2021.

Sapova, M. D. and Fedorov, A. K. Variational quantum eigensolver techniques for simulating carbon monoxide oxidation. *Communications Physics*, 5(1):1–13, 2022.

Schatzki, L., Arrasmith, A., Coles, P. J., and Cerezo, M. Entangled datasets for quantum machine learning. *arXiv preprint arXiv:2109.03400*, 2021.

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

Shor, P. W. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134. Ieee, 1994.

Shor, P. W. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

Wang, H., Ding, Y., Gu, J., Li, Z., Lin, Y., Pan, D. Z., Chong, F. T., and Han, S. Quantumnas: Noise-adaptive search for robust quantum circuits. In *The 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA-28)*, 2022a.

Wang, H., Ding, Y., Gu, J., Lin, Y., Pan, D. Z., Chong, F. T., and Han, S. Quantumnas: Noise-adaptive search for robust quantum circuits. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 692–708. IEEE, 2022b.

Wang, P.-Y., Usman, M., Parampalli, U., Hollenberg, L. C., and Myers, C. R. Automated quantum circuit design with nested monte carlo tree search. *arXiv preprint arXiv:2207.00132*, 2022c.

Wang, Z., Liang, Z., Zhou, S., Ding, C., Shi, Y., and Jiang, W. Exploration of quantum neural architecture by mixing quantum neuron designs. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–7. IEEE, 2021.

West, D. B. et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

Williams, C. P. and Gray, A. G. Automated design of quantum circuits. In Williams, C. P. (ed.), *Quantum Computing and Quantum Communications*, pp. 113–125, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-49208-5.

Wu, J., Liu, Y., Zhang, B., Jin, X., Wang, Y., Wang, H., and Yang, X. A benchmark test of boson sampling on tianhe-2 supercomputer. *National Science Review*, 5(5): 715–720, 2018.

Wu, Y., Bao, W.-S., Cao, S., Chen, F., Chen, M.-C., Chen, X., Chung, T.-H., Deng, H., Du, Y., Fan, D., et al. Strong quantum computational advantage using a superconducting quantum processor. *Physical review letters*, 127(18): 180501, 2021.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., and Xiong, H. Pc-darts: Partial channel connections for memory-efficient architecture search. *arXiv preprint arXiv:1907.05737*, 2019.

Zeng, Y., Zhou, Z.-Y., Rinaldi, E., Gneiting, C., and Nori, F. Approximate autonomous quantum error correction with reinforcement learning. *arXiv preprint arXiv:2212.11651*, 2022.

Zhang, S.-X., Hsieh, C.-Y., Zhang, S., and Yao, H. Neural predictor based quantum architecture search. *Machine Learning: Science and Technology*, 2(4):045027, 2021.

Zhang, S.-X., Hsieh, C.-Y., Zhang, S., and Yao, H. Differentiable quantum architecture search. *Quantum Science and Technology*, 7(4):045023, 2022.

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

A. Detailed Description of Baselines

Evolutionary algorithms mainly include heuristic search approaches (Rigby, 2021; Ding & Spector, 2022; Li et al., 2017; Deibuk & Biloshytskyi, 2015; Lamata et al., 2018; Ding et al., 2006; Williams & Gray, 1999) where genetic algorithms account for a large part. In a genetic algorithm for QAS, the chromosome of a population is usually encoded by the type of quantum gate and the wires to which it is connected. The fitness function is defined by the tasks to be completed, measuring the quality of a chromosome, which can be the correctness of calculation result for a quantum adder (Deibuk & Biloshytskyi, 2015), or the distance value of the searched and target unitary (Williams & Gray, 1999) for a unitary approximation task. Common operations over chromosomes such as crossover, and mutation can also be utilized in QAS algorithms.

Evolutionary algorithms are always limited to the design of QCs without rotation gates. For PQCs, the searching approaches usually fit the following two-step paradigm: searching for a gate arrangement (architecture parameters), and optimizing the parameters within the gates (rotation parameters). The design of PQCs requires optimizing architecture and rotation parameters iteratively.

Differentiable search methods can optimize both architecture and rotation parameters in an end-to-end differentiable manner. There are still relatively few works based on this method because it is difficult to design gradients for architecture parameters. One possible solution is to approximate the gradient by sampling. A Monte Carlo gradient is designed in (Zhang et al., 2022) to update the architecture parameters, and rotation parameters can be tackled by simple gradient descent. However, sampling-based methods often face the problem of low sample efficiency, which may lead to slow search speed and high resource consumption.

Reinforcement learning (RL) methods involve teaching an agent to take suitable actions to maximize the total reward in a particular environment (Nautrup et al., 2019; Ostaszewski et al., 2021). In the QAS setting, the action taken by an agent is to decide which gate to choose and which position to take. The reward is specified by different tasks, evaluating the actions taken by the agent. Different RL methods can be used for QAS. (Ostaszewski et al., 2021) leverages Double Deep-Q network (DDQN) (Mnih et al., 2013), and (Kuo et al., 2021) makes use of Proximal Policy Optimization (PPO) (Schulman et al., 2017). (Wang et al., 2022c) leverages Monte Carlo Tree Search (MCTS) for the optimization of architecture parameters and update rotation parameters by gradient descent.

Hybrid methods include all the other QAS approaches which follow the above-mentioned paradigm that optimizes architecture and rotation parameters iteratively. For hybrid algorithms there may be several optimization methods for architecture parameters. (Grimsley et al., 2019; Sapova & Fedorov, 2022) calculate the gradient of candidate operators at each searching step to decide which operators to choose next. (Zhang et al., 2021) trains a neural network to choose better candidate gates for future optimization. (Du et al., 2022) randomly samples architecture parameters, iterates the rotation parameters. It leverages a genetic algorithm to rank the structures then finally finetune the circuit found.

Apart from the above four categories of baseline in QAS, we also mention the works on neural architecture search (NAS) on classic computers, which is a booming research field in recent years. In general it aims to automatically search for high-performance neural networks for any given task. Genetic algorithm (Real et al., 2017; 2019; Liu et al., 2021) and RL (Zoph & Le, 2016; Jaafra et al., 2019) are two promising methods for NAS. The emergence of DARTS (Liu et al., 2018b) relaxed the searching of discrete neural network components into a continuous domain and put forward a differentiable searching paradigm for NAS, boosting following works such as (Liang et al., 2019; Casale et al., 2019; Xu et al., 2019). However, the search process of DARTS contains parallel computation of each network component, which takes up a lot of memory. Various methods have been proposed such as (Dong & Yang, 2019), which leverages Gumbel-Softmax (Jang et al., 2016) to reduce computation and (Casale et al., 2019), which views the architecture parameters as probabilistic parameters and sample certain subnet components for evaluation each time.

B. Technical Implementation Details of Baseline Algorithms

Six representative QAS baselines are introduced. We implement/change all the code into PyTorch version.

B.1. Brute Force Search

We design a naive BFS for both two datasets and add a bidirectional version for QC Regeneration dataset.

For brute force search algorithm, we want to find M gates, whose matrices are $\boldsymbol{A}_0, \cdots, \boldsymbol{A}_{M-1}$ respectively, such that

$$\prod_{i=0}^{M-1} \boldsymbol{A}_i = \boldsymbol{U}_t. \tag{10}$$

The naive brute force search method is to sequentially enumerate all possible M and $\boldsymbol{A}_i$ until we find an answer. This algorithm tends to take up a significant amount of time and space. To optimize the naive method, we come up with two approaches, which can be further combined.

One approach is to use dynamic programming to remove duplicated matrices at each step.

Lemma B.1. *We can use dynamic programming in the brute force search algorithm to reduce time complexity.*

Proof. Assume $\mathcal{A}_m$ represents the set of all possible circuits with m gates. If we randomly pick two matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathcal{A}_1$, then in some cases $\boldsymbol{AB} = \boldsymbol{BA}$. For this reason, $|\mathcal{A}_m|$ is actually smaller than $|\mathcal{A}_1|^m$. More generally, $|\mathcal{A}_{m_1+m_2}| \leq |\mathcal{A}_{m_1}||\mathcal{A}_{m_2}|$. For example, if $\boldsymbol{A}_0\boldsymbol{A}_1 = \boldsymbol{A}_1\boldsymbol{A}_0$, then $\boldsymbol{A}_0\boldsymbol{A}_1\boldsymbol{A}_2\cdots\boldsymbol{A}_{m-1} = \boldsymbol{A}_1\boldsymbol{A}_0\boldsymbol{A}_2\cdots\boldsymbol{A}_{m-1}$.

When we use $\mathcal{A}_m = \mathcal{A}_{m-1}\mathcal{A}_1$ to calculate $\mathcal{A}_m$ for all $m \leq M$ recursively, we can remove the duplicate matrices at each step, and the time needed to remove duplicates T_{remove} is relatively shorter than the time needed to calculate results of matrix multiplication. Now the time complexity of the calculation of $\mathcal{A}_m$ is $T(\mathcal{A}_m) = T(\mathcal{A}_{m-1}) + T_{\text{mul}}|\mathcal{A}_{m-1}||\mathcal{A}_1| + T_{\text{remove}} \approx T_{\text{mul}}|\mathcal{A}_{m-1}||\mathcal{A}_1|$, which is significantly shorter than the original time $T_{\text{mul}}|\mathcal{A}_1|^m$, where T_{mul} is the time one matrix multiplication operation takes.

□

Another approach is to use bidirectional brute force search.

Notice that when $M \geq 2$, Eq. 10 is equivalent to

$$\prod_{i=0}^{\lfloor M/2 \rfloor - 1} \boldsymbol{A}_i = \boldsymbol{U}_t \Big(\prod_{i=\lfloor M/2 \rfloor}^{M-1} \boldsymbol{A}_i \Big)^{-1} \tag{11}$$

Note that $\boldsymbol{A} \in \mathcal{A}_{\lfloor M/2 \rfloor} \cap \boldsymbol{U}_t\mathcal{A}^{-1}_{\lceil M/2 \rceil}$ and the problem is solved. The brute force search is listed in Alg. 1. Concatenation of circuit $\mathbf{QC}_1$ and $\mathbf{QC}_2$ means for a blank circuit, we first add gates $g \in \mathbf{QC}_1$ sequentially to the circuit, then add gates $g \in \mathbf{QC}_2$ sequentially to the circuit. The time complexity is now only $O(|\mathcal{A}_{\lceil M/2 \rceil}|T_{\text{mul}})$, with a squared speedup.

Algorithm 1 Bidirectional Brute Force Search for QC Regeneration

Input $\boldsymbol{U_t}$
  Initialize $M = 1, \mathcal{A}_0 = \{\sigma(I)\}, \mathcal{A}_1 = \{\sigma(g)|g \in \mathcal{G}\}$
  while $\mathcal{A}_{\lfloor M/2 \rfloor} \bigcap \boldsymbol{U_t}\mathcal{A}^{-1}_{\lceil M/2 \rceil} = \emptyset$ **do**
    1. $M = M + 1$
    2. Calculate $\mathcal{A}_{\lfloor M/2 \rfloor}$ and $\mathcal{A}_{\lceil M/2 \rceil}$ respectively by dynamic programming
    3. Calculate $\boldsymbol{U}_t\mathcal{A}^{-1}_{\lceil M/2 \rceil}$
  end while
  Find $\forall \boldsymbol{A} \in \mathcal{A}_{\lfloor M/2 \rfloor} \bigcap \boldsymbol{U}_t\mathcal{A}^{-1}_{\lceil M/2 \rceil}$, where $\boldsymbol{A}$ represents circuit $\mathbf{QC}_1$ in $\mathcal{A}_{\lfloor M/2 \rfloor}$ and circuit $\mathbf{QC}_2$ in $\boldsymbol{U}_t\mathcal{A}^{-1}_{\lceil M/2 \rceil}$
  Return concatenate $\mathbf{QC}_1$ and $\mathbf{QC}_2$.

As for the naive brute force search to solve the QC Regeneration and Unitary Approximation datasets, we only need to enumerate m and find $\boldsymbol{A} \in \mathcal{A}_m$ that minimize Eq. 2.

B.2. Simulated Annealing

The initial circuit is filled with I gates. For the t-th step, we randomly pick i, replace $\boldsymbol{A}_i$ with the unitary matrix of a random gate, and calculate the change of L, $\Delta L_t = L_t - L_{t-1}$. If $\Delta L_t < 0$, we accept this change, else we accept this change at probability $e^{-\Delta L_t/T_t}$, where $T_t = \alpha T_{t-1}$, T_0 is called the initial temperature parameter, T_0 and α are both changeable hyperparameters. We repeat this process until the loss remains stable for a few iterations or the maximum iteration limit is achieved, then $\boldsymbol{A}_0, \cdots, \boldsymbol{A}_{M-1}$ are submitted as our solution.

B.3. Genetic Algorithm

Following (Williams & Gray, 1999), the quantum gates can be encoded by the quantum gate type and the qubit to which it is connected. When searching for a certain quantum layout, gates are laid sequentially to form a chromosome. Implementation details are not explained in the original paper such as the management method of the population and the code is not open-sourced. We also refer to (Prins, 2004) for the concrete design of the genetic algorithm, which is a classical work for designing an effective genetic algorithm for the vehicle routing problem (VRP).

The evaluation metric of genetic algorithm follows Eq. 2, which can serve as the fitness function.

Ranking-Based Scheme and Selection Probability Distribution are proposed in (Williams & Gray, 1999) for the selection of children generation. However, there is something wrong with the equation of the Selection Probability Distribution method and the Ranking-Based Scheme can be substituted with more effective roulette wheel selection and tournament algorithms. In our algorithm, we use the roulette wheel selection algorithm for choosing the children generation.

Search operators are needed for the generation of new chromosomes. The mutation, substitution, crossover, transposition, insertion, and deletion operations are implemented in our baseline genetic algorithm.

After generating enough children chromosomes, generation management is required to replace some items in the parent generation with the newly generated chromosomes. In the baseline genetic algorithm of our benchmark, to compose the next generation, parent chromosomes are sorted by fitness value, then the first half of chromosomes are kept. The second half of parent chromosomes are substituted by children chromosomes using wheel selection. This method can maintain population diversity, and alleviate the situation of rapidly converging to a locally optimal solution.

B.4. Hybrid Algorithm

The hybrid algorithm in (Du et al., 2022) combines gradient-based search with a classical search algorithm (genetic algorithm). It can make use of $\mathcal{G}_R$ as it is able to optimize the rotation angle parameters by gradient propagation.

Hybrid algorithm embeds quantum architecture search into an image classification over MNIST and decomposes QC into layers. It divides the searching process into 2 steps. In the first step, it randomly samples ansatze from the pre-defined ansatz pool for each layer. Then for each ansatz sampled, it optimizes the rotation angle parameters by autograd function provided by PennyLane (Bergholm et al., 2018) for a few steps (20 in our setting). In the second step, it ranks the optimized ansatze (leveraging genetic algorithm) and finetunes the rotation parameters of the best structure.

The original hybrid algorithm can not be directly used for Unitary Approximation. To qualify hybrid algorithm for the dataset, we make a few revisions to the algorithm. Firstly, as genetic algorithm has been listed above as one baseline algorithm, we remove the genetic part of the algorithm to compare these algorithms more fairly. Secondly, we get rid of the restrictions over the pre-defined ansatz pool. The QC supernet of each layer is now composed of both single-qubit quantum gates (Rotation gates) and multi-qubit quantum gates (CNOT gates). For a circuit with n qubits, the single-qubit quantum gate set $\mathcal{S}$ and multi-qubit quantum gate set $\mathcal{M}$ can be calculated by

$$\mathcal{S} = R_0 \times R_1 \times \cdots R_k \times \cdots R_{n-1} \tag{12}$$

$$\mathcal{M} = P(\{\mathrm{CNOT}_{i,j} | \forall i, j \in [0, n-1], i \neq j\}) \tag{13}$$

where $\times$ means Cartesian product, P means the operation that calculates the power set. $R_k = \{R_x(\theta)_k, R_y(\theta)_k, R_z(\theta)_k\}$, k denotes the qubit the gate is on. i, j denote the control qubit and the target qubit of CNOT gate respectively. But this method will lead to a huge amount of CNOT gates. To alleviate such situation, we reduce the size of $\mathcal{M}$ by only allowing control bit and target bit in adjacent wires and the total ansatz pool is the Cartesian product of $\mathcal{S}$ and $\mathcal{M}$. The third revision is changing the loss function to Eq. 2. Then the algorithm can be tested for Unitary Approximation dataset.

We also test hybrid algorithm's performance on Unitary Approximation using $\mathcal{G}_C$. Under this circumstance, the optimization of rotation gate angles is removed. The single-qubit gate set $\mathcal{S}$ is changed to

$$\mathcal{S} = \mathcal{G}_{S_0} \times \mathcal{G}_{S_1} \times \cdots \mathcal{G}_{S_k} \times \cdots \mathcal{G}_{S_{n-1}} \tag{14}$$

$\mathcal{G}_{S_k} = \{H_k, S_k, T_k, I_k\}$, k denotes the qubit the gate is on. The dataset of QC Regeneration is also tested under this setting.

B.5. Differentiable Algorithm

The differentiable algorithm (Zhang et al., 2022) is similar with the hybrid algorithm. HA chooses the architecture parameters by random sampling, whereas DA designs a Monte Carlo gradient for structure sampling and calculates it by a mean-field probability model. For a n-qubit m-layer quantum circuit, DA chooses gates for each position of the $O(m \times n)$ space.

We re-implement DA by the tutorial of TensorCircuit [2]. We make a few revisions. For the k-th qubit, the gates to choose for $\mathcal{G}_R$ is $R_k \cup M_k$, where $R_k = \{R_x(\theta)_k, R_y(\theta)_k, R_z(\theta)_k\}$, $M_k = \{\mathrm{CNOT}_{i,k} | \forall i \in [0, n-1], i \neq k\}$. The gates to choose for $\mathcal{G}_C$ is $\mathcal{G}_{S_k} \cup M_k$, where $\mathcal{G}_{S_k} = \{H_k, S_k, T_k, I_k\}$. For rotation gate parameters, they are updated by simple gradient propagation, for architecture parameters they are updated by a proposed Monte Carlo gradient using mean field theory. The loss of the network is shifted to L of the searched matrix and the target matrix which can be calculated by Eq. 2.

However, we find the Monte Carlo method just samples with very low efficiency. It even fails to achieve reasonable results for the 3-qubit case, and searching for a 5-qubit circuit in our setting will take up more than 4 hours. The tutorial of TensorCircuit suggests that we can just use Softmax in the simulator (as there is no restriction for a unitary matrix during the intermediate searching steps) and calculate the gradient for quicker speed and higher efficiency. Thus in our evaluating code, for QC Generation task we test with Monte Carlo gradient, for Unitary Approximation task we test 2-qubit 3-qubit tasks with Monte Carlo gradient and 4-qubit 5-qubit tasks with Softmax over architecture parameters.

B.6. Reinforcement Learning

Recently, reinforcement learning algorithms have also started to be used for quantum searching datasets to construct VQE circuits (Ostaszewski et al., 2021). In our baselines, we also adapt it to an algorithm that can accomplish both QC Regeneration and Unitary Approximation.

The original RL algorithm for VQE employs a DDQN (Mnih et al., 2013) with $\epsilon-$greedy policy and an Adam optimizer. The agent takes each action by adding a new gate from the candidate set $\mathcal{G}_R$ on a certain qubit of the current circuit.

A QC is encoded into a list containing tuples of 4 elements, each tuple encodes the type and the position of gates. The first two elements of the tuple represent rotation gates, where the first element means which qubit the gate is on and the second element means the type of the rotation gate (1: $R_x(\theta)$, 2: $R_y(\theta)$, 3: $R_z(\theta)$). The last 2 elements encode CNOT gates. The third and fourth elements represent the control qubit and target qubit respectively. If the gate is a rotation gate, another rotation angle parameter θ is set.

The original RL algorithm is designed for VQE problem which optimizes θ for each rotation gate to minimize the energy at each state of one episode. We adjust it to adapt to our quantum architecture search datasets. The aim of our dataset is to find the QC that minimizes L, so we define the energy by matrix distance L which can be calculated by Eq. 2. In this case, minimizing the ground state energy changes into minimizing L.

We also adjust the origin reward function to

$$R = \begin{cases} 5 & E_t < \epsilon \\ -5 & E_t > \epsilon \text{ and } t > M \\ \mathrm{clip}(\frac{E_{t-1} - E_t}{\max(E_{t-1}, 1)}, -1, 1) & \text{O.W.} \end{cases} \tag{15}$$

where E_t means the energy after step t. At each step, the agent lays a gate according to the policy, so t also represents the total gate number after step t. ϵ is a parameter generated by a feedback-driven curriculum learning method to make energy fall steadily. If $E_t < \epsilon$, the agent will obtain 5 reward, which means we have found a reasonable circuit. If $E_t > \epsilon$, and $t > M$ which is the maximum number of gates we have set, the agent will obtain -5 reward. E_{t-1} is the energy of the previous step. If step t leads to a smaller energy, the agent will get a positive reward.

We also test RL's performance on QC Regeneration and Unitary Approximation by using $\mathcal{G}_C$ as the action candidate set. Because $\mathcal{G}_C$ has no parameter θ, there is no need to optimize the rotation angle parameter, but we have to change the encoding policy. For the tuple of 4 elements, the gates occupying only one qubit are encoded by the first two elements of the tuple, where the first element means the qubit position of the gate, and the second element means the type of gates (1: H, 2: S, 3: T, 4: I).The encoding scheme of CNOT gates is the same as above.

[2] https://tensorcircuit.readthedocs.io/en/latest/tutorials/dqas.html

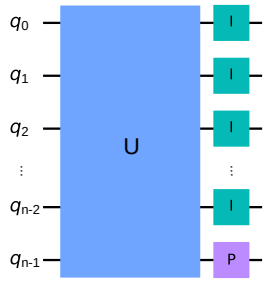


Figure 7: QC for an arbitrary unitary

C. Lemma and Proof

Lemma C.1. *By leveraging $\mathcal{G}_R$ we can only generate circuits whose matrix determinant is ± 1.*

Proof. In the Unitary Approximation dataset, we use $\mathcal{G}_R = \{R_x(\theta), R_y(\theta), R_z(\theta), \mathrm{CNOT}\}$ to approximate arbitrary unitary matrices. Where

$$R_P(\theta) = \exp(-\mathrm{i}\theta P/2) = \cos(\theta/2)I - \mathrm{i}\sin(\theta/2)P \tag{16}$$

I is the identity gate and $P = \{X, Y, Z\}$, which is called Pauli rotation. Under this definition,

$$\boldsymbol{R_x}(\theta) = \begin{pmatrix} \cos(\theta/2) & -\mathrm{i}\sin(\theta/2) \\ -\mathrm{i}\sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \boldsymbol{R_y}(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \boldsymbol{R_z}(\phi) = \begin{pmatrix} \exp(-\mathrm{i}\phi/2) & 0 \\ 0 & \exp(\mathrm{i}\phi/2) \end{pmatrix}$$

and for a CNOT gate with 0-bit as the control bit and 1-bit as the target bit, $\mathbf{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.

Assume $|\boldsymbol{A}|$ calculates the determinant of a matrix $\boldsymbol{A}$, we notice that $|\boldsymbol{R_x}(\theta)| = |\boldsymbol{R_y}(\theta)| = |\boldsymbol{R_z}(\theta)| = 1$, and $|\mathbf{CNOT}| = -1$. In the construction of quantum gates for QCs, we just use two operations: tensor product (Kronecker product) and matrix multiplication.

For tensor product, we have $|\boldsymbol{X} \bigotimes \boldsymbol{Y}| = |\boldsymbol{X}|^n \cdot |\boldsymbol{Y}|^m, \boldsymbol{X} \in \mathbb{C}^{m\times m}, \boldsymbol{Y} \in \mathbb{C}^{n\times n}$, and for matrix multiplication, we have $|\boldsymbol{XY}| = |\boldsymbol{X}||\boldsymbol{Y}|, \boldsymbol{X} \in \mathbb{C}^{n\times n}, \boldsymbol{Y} \in \mathbb{C}^{n\times n}$. If we use $\mathcal{G}_R$, the determinants of whose matrices are all ± 1 to construct QCs, the determinant of the final circuit will be kept to ± 1. In this case, we can only generate QCs whose determinants are ± 1.

□

Lemma C.2. *For a target unitary matrix with a determinant not equal to 1, we can construct a new matrix whose determinant is 1, and the QC represented by the new matrix will have no difference with that of the target matrix when making an observation of the output quantum state.*

Proof. In Fig. 7, for a QC with n qubits, assume the unitary to approximate is $\boldsymbol{U}$, and $|\boldsymbol{U}| = a + b\mathrm{i}$, (s.t. $a^2 + b^2 = 1$). In this case, we add a layer of gates to the circuit represented by $\boldsymbol{U}$. For qubit from 0 to $n-2$, we add I gates and for qubit $n-1$, we add a phase gate (P gate).

Firstly, we prove that the addition of this layer of gates will not influence the possibility of each state being measured. Assume a state vector of $|q\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$, (s.t. $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$) after $\boldsymbol{U}$, and the rotation angle of gate P is ϕ. The unitary of the last layer can be calculated by $\boldsymbol{U_{-1}} = \boldsymbol{I}^{n-1} \bigotimes \boldsymbol{P}$, where $\boldsymbol{I}$ is the unitary matrix of I gate (i.e. $\boldsymbol{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$), and $\boldsymbol{P}$ is the matrix of P gate, which is $\boldsymbol{P}(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & \exp(\mathrm{i}\phi) \end{pmatrix}$ for a phase gate.

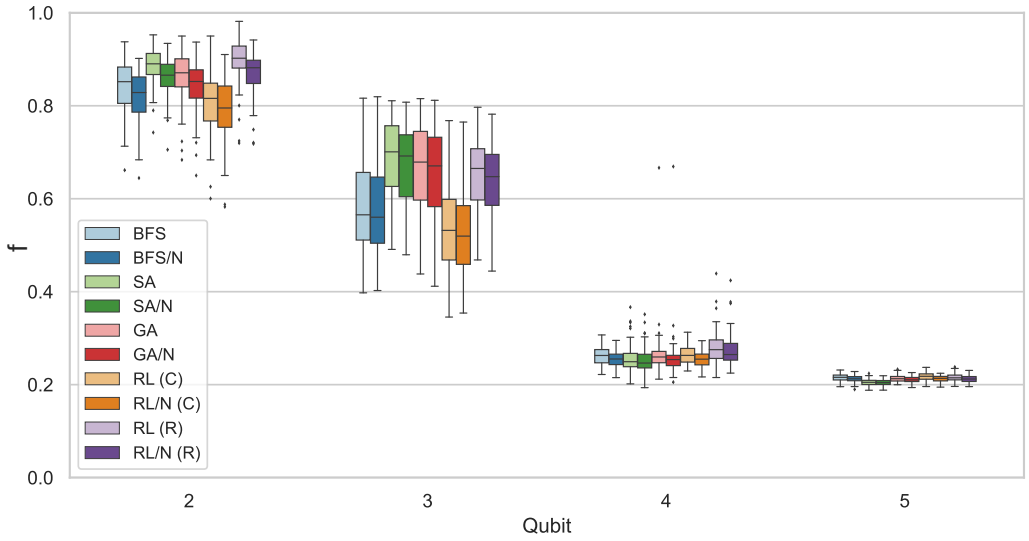


Figure 8: Box-plot results of circuit with noise (BFS, SA, GA, RL)

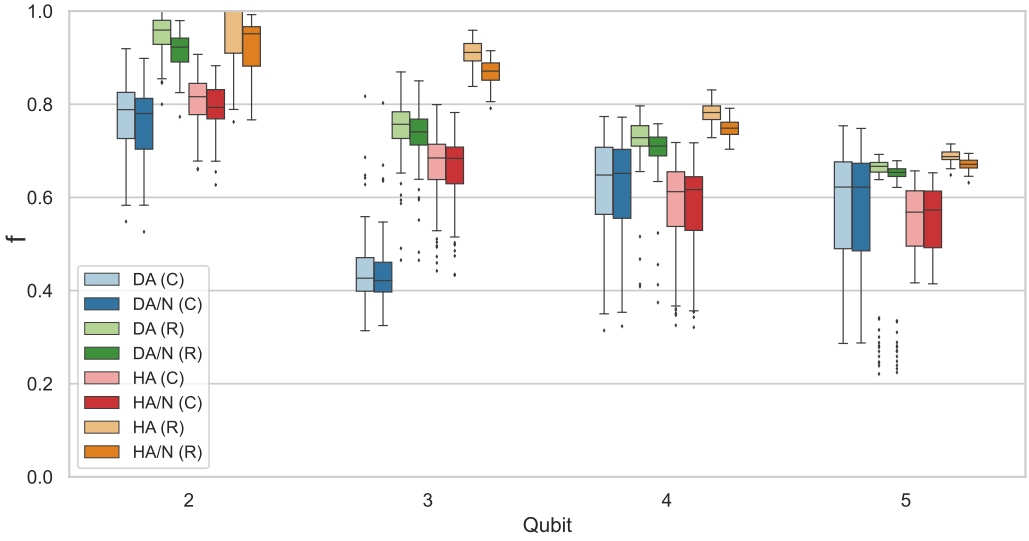


Figure 9: Box-plot results of circuit with noise (DA, HA)

Calculating $\boldsymbol{U_{-1}}$ will result in a matrix with elements on the diagonal. The elements on the diagonal are $(1, \exp(\mathrm{i}\phi), 1, \exp(\mathrm{i}\phi), \cdots, 1, \exp(\mathrm{i}\phi))$. Passing state $|q\rangle$ through $\boldsymbol{U_{-1}}$ will get $|q'\rangle = \boldsymbol{U_{-1}}|q\rangle = \sum_{i=0}^{2^{n-1}-1} \alpha_{2i}|2i\rangle + \sum_{i=0}^{2^{n-1}-1} \exp(\mathrm{i}\phi) \cdot \alpha_{2i+1}|2i+1\rangle$. When making an observation of the output quantum state, the possibility of observing $|2i\rangle$ is $\alpha_{2i} \cdot \bar{\alpha_{2i}}$, observing $|2i+1\rangle$ is $\exp(\mathrm{i}\phi)\alpha_{2i+1} \cdot \exp(-\mathrm{i}\phi)\bar{\alpha_{2i+1}} = \alpha_{2i+1} \cdot \bar{\alpha_{2i+1}}$, which will not result in a difference.

Secondly, we need to calculate the determinant of the new matrix $\boldsymbol{U'} = \boldsymbol{U}\boldsymbol{U_{-1}}$. We already have $|\boldsymbol{U_{-1}}| = \exp(\mathrm{i}2^{n-1}\phi) = \cos(2^{n-1}\phi) + \mathrm{i}\sin(2^{n-1}\phi)$ and $|\boldsymbol{U}| = a + b\mathrm{i}$, (s.t. $a^2 + b^2 = 1$). We only need to match the determinant of $\boldsymbol{U_{-1}}$ with that of $\boldsymbol{U}$. To be more specific, let $\cos(2^{n-1}\phi) = a$ and $\sin(2^{n-1}\phi) = -b$, we solve the equations and the ϕ is set.

Following the above two steps, we can get a new matrix whose determinant is 1, and the QC represented by the new matrix will have no difference from that of the target matrix when making an observation of the output quantum state. □

D. Results With Circuit Noise

We test 6 baseline algorithms on the Unitary Approximation dataset under a 4.77% readout circuit noise. We add a bit flip at the end of the circuit to simulate the readout error. The error rate is obtained from two superconducting quantum devices sycamore (Arute et al., 2019) and zuchongzhi (Wu et al., 2021). The results are showed in Fig. 8 and Fig. 9. In the two figures, **(Model)/N** means we test the circuits under readout noise. All the algorithms show performance degradation in noisy environments.