Optimizing Quantum Circuit Placement via Machine Learning

Hongxiang Fan
Department of Computing
Imperial College London
London, UK
h.fan17@imperial.ac.uk

Ce Guo
Department of Computing
Imperial College London
London, UK
c.guo@imperial.ac.uk

Wayne Luk
Department of Computing
Imperial College London
London, UK
w.luk@imperial.ac.uk

ABSTRACT

Quantum circuit placement (QCP) is the process of mapping the synthesized logical quantum programs on physical quantum machines, which introduces additional SWAP gates and affects the performance of quantum circuits. Nevertheless, determining the minimal number of SWAP gates has been demonstrated to be an $\mathcal{NP}$-complete problem. Various heuristic approaches have been proposed to address QCP, but they suffer from suboptimality due to the lack of exploration. Although exact approaches can achieve higher optimality, they are not scalable for large quantum circuits due to the massive design space and expensive runtime. By formulating QCP as a bilevel optimization problem, this paper proposes a novel machine learning (ML)-based framework to tackle this challenge. To address the lower-level combinatorial optimization problem, we adopt a policy-based deep reinforcement learning (DRL) algorithm with knowledge transfer to enable the generalization ability of our framework. An evolutionary algorithm is then deployed to solve the upper-level discrete search problem, which optimizes the initial mapping with a lower SWAP cost. The proposed ML-based approach provides a new paradigm to overcome the drawbacks in both traditional heuristic and exact approaches while enabling the exploration of optimality-runtime trade-off. Compared with the leading heuristic approaches, our ML-based method significantly reduces the SWAP cost by up to 100%. In comparison with the leading exact search, our proposed algorithm achieves the same level of optimality while reducing the runtime cost by up to 40 times.

KEYWORDS

Quantum circuit placement, Machine learning, Deep reinforcement learning, Evolutionary algorithm

1 INTRODUCTION

Quantum Computing (QC) has attracted significant research and industrial interest due to its potential in various applications such as data security [26] or quantum chemistry [4]. After IBM released its first publicly available quantum processor [12], Intel and Google also announced their 49 and 72 qubits quantum systems [10, 11] to join the quantum race. Although quantum hardware is becoming more sophisticated, the efficiency and scalability of software tools for QC devices has been challenged due to the increasing number of qubits and the growing size of quantum circuits [13, 35].

There are two processes involved in implementing a quantum program on an actual quantum device: logic synthesis and quantum circuit placement (QCP). Logic synthesis receives a quantum program as input, and translates it into a list of primitive gates supported by the underlying library. Then, QCP is applied to map the logically-synthesized quantum circuit on the physical quantum devices. As the underlying quantum hardware may have various constraints such as the limited qubit connections, several extra SWAP gates may be required by QCP, which increases the size of quantum circuits and harms the fidelity of the execution [35]. Nevertheless, determining the minimal number of SWAP gates for a quantum circuit on a quantum device has been demonstrated to be an $\mathcal{NP}$-complete problem [27], confirming the difficulty of QCP.

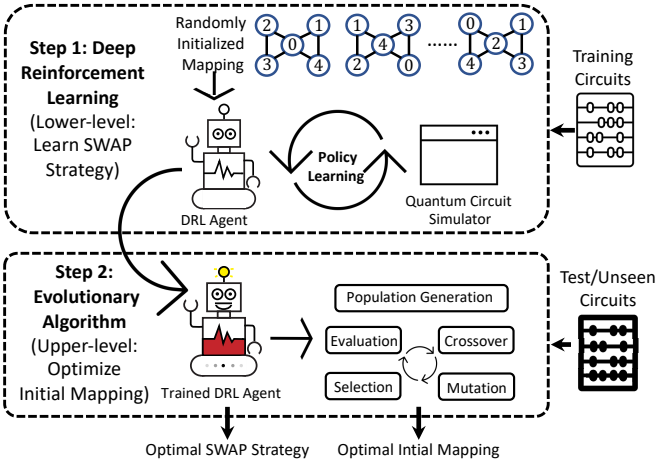

Figure 1: An overview of our ML-based approach.

Various exact methods have been proposed to address QCP [5, 29, 34, 35], but they suffer from poor scalability on large quantum circuits. For instance, the leading exact method OLSQ-SWAP [29] takes nearly half an hour on a high-end CPU server to map a 8-qubit quantum adder *vbe_adder_3* on an IBM Melbourne quantum machine. Moreover, the increasing number of qubits in recently-released quantum devices and the growing size of quantum circuits further exaggerate the performance of exact approaches. Although various heuristic approaches have been proposed to speedup QCP, Tan *et al.* [30] suggest that there is still a huge optimality gap between the exact solutions and the leading heuristic methods [3, 13, 28]. For example, the leading heuristic approach t|ket⟩ [28] needs 38 SWAP gates for *queko_15_1* quantum circuit on Rigetti Aspen-4 quantum machine, but the exact search [29] requires no SWAP cost. The expensive runtime cost of exact methods and the sub-optimal issue of heuristic solutions indicate a great demand for a better approach to tackle QCP.

In this paper, we propose a machine learning (ML)-based framework to address QCP. An overview of our proposed ML-based approach is illustrated in Figure 1. We formulate QCP as a bilevel optimization problem. As the upper-level problem of optimizing the initial qubit mapping contains a vast design space, we adopt an evolutionary algorithm to minimize the SWAP cost, which also

offers users the flexible exploration of optimality-runtime trade-off. Then, we observe that the lower-level problem is essentially a combinatorial optimization problem of minimizing the SWAP cost given an initial qubit mapping. To enable the generalization ability of our framework, we adopt a policy-based deep reinforcement learning (DRL) algorithm with the capability of knowledge transfer. To address the suboptimality caused by gate-by-gate or level-by-level processing [30], we encode the whole circuit's information into our state space to optimize the SWAP strategy.

Our contributions in this paper can be summarized as follows:

- A novel ML-based framework to address quantum circuit placement (QCP), which adopts an evolutionary algorithm (EA) in the upper level to optimize the initial mapping. The EA-based method also offers the flexibility for users to explore the optimality-runtime trade-off (Section 3.1 & 3.3).
- Optimizing the SWAP strategy using a policy-based deep reinforcement learning (DRL) agent, which decreases the SWAP cost by encoding the whole circuit's information into state space. The generalization ability of our framework is improved by using the knowledge transfer of our DRL agent (Section 3.2).
- Extensive experiments on a wide range of quantum circuits demonstrate the effectiveness of our ML-based framework compared with heuristic and exact methods in terms of both optimality and runtime cost (Section 4).

2 BACKGROUND

2.1 Quantum Circuit and QCP

The quantum circuits specified by quantum programs contain a list of quantum gates to perform quantum computation. The common quantum gates include the single-qubit X gate that negates the qubit, and the two-qubit CX that only changes the second qubit by XOR the first and second qubits. In this paper, we assume the quantum circuits have been processed by logic synthesis, so the circuits only contain primitive gates supported by the underlying library. Using Qiskit library as an example, a logically-synthesised quantum circuit is shown in Figure 2(a), which only includes primitive quantum gates such as X, CX, S, T and $T^\dagger$. The horizontal lines $l_{0\sim4}$ represent the logic qubits. The two-qubit gates are specified by vertical lines indicating their connection relationships.

The process of QCP is to map the logic quantum circuits on the physical quantum devices such as IBM QX2 (Figure 2(b)). The main difficulty during the QCP is that the physical qubit connections sometimes cannot meet the connective requirements of two-qubit quantum gates. For instance, in order to execute g_1 and g_2 in Figure 2(c), it requires the physical connections on qubit pairs (l_0, l_2) and (l_1, l_3). Therefore, we initially map logical qutbits $l_{0\sim4}$ to the physical qubits p_2, p_3, p_0, p_4 and p_1 respectively. However, while executing g_3, the physical connection on p_0 (l_2) and p_4 (l_3) is not available using our initial mapping, making g_3 inexecutable. To address this issue, the common practice is to insert a SWAP gate [35] between l_0 and l_3 to exchange the physical qubits p_2 and p_4. This SWAP insertion makes g_3 executable as its physical connection are available under the new qubit mapping. Nevertheless, the extra SWAP gates may harm the performance of quantum circuits. Therefore, one of the main objectives for QCP is to decrease the number

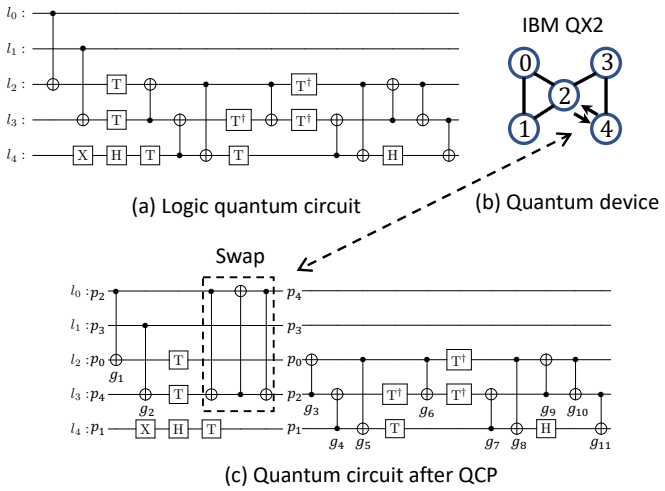

Figure 2: A quantum circuit after logic synthesis.

of SWAP gates [30]. It has been demonstrated that determining the minimal numberof SWAP gates is an $\mathcal{NP}$-complete problem [27].

2.2 Related Work

2.2.1 Exact and Heuristic Approaches for Quantum Circuit Placement. To minimize the number of SWAP gates required by QCP, various exact methods have been proposed. Shafaei *et al.* [25] provided a routing algorithm by solving QCP as Minimum Linear Arrangement problem. By formulating the QCP problem as a symbolic optimization problem, Wille *et al.* [34, 35] adopted a Boolean satisfiability solver to minimize the extra gate cost. Bhattacharjee *et al.* [5] used a multi-tier method to address QCP via multiple levels in parallel. Tan *et al.* [29, 30] found that the extra constraints were imposed in previous exact methods by using the gate-by-gate or the level-by-level arrangement. By changing the formulation, their work improved the optimality of QCP while reducing the runtime cost. However, as the number of quantum bits (qubit) becomes larger, the runtime of these exact methods increases exponentially, making them not feasible in the near future.

To speed up QCP, various heuristic approaches have been proposed. Siraichi *et al.* [27] optimized the initial mapping based on the out-degrees of the graphs constructed by quantum circuits. Then, the SWAP gates were inserted according to the heuristic distance between different qubits. Zulehner [38] addressed the QCP problem for IBM's QX architecture using an A^* search algorithm together with a look-ahead scheme. Childs *et al.* [6] used routing via matchings and token swapping frameworks to minimize the number of SWAP gates required. Li *et al.* [13] tackled QCP using a SWAP-based bidirectional heuristic search algorithm with a reverse traversal technique to optimize the initial mapping. Tannu *et al.* [31] and Murali *et al.* [18] also considered the gate fidelity into QCP. Sivarajah [28] provided an open-source framework t|ket⟩ [28], which showed leading results among various heuristic approaches. However, the optimality gap between heuristic and exact approaches [30] suggests an urgent need for a better solution.

2.2.2 ML Algorithms for Quantum Circuit Design. There has been an increasing interest in applying ML algorithms for QC, but most

of the work only focused on optimizing logical quantum circuits instead of studying QCP. Targeting on variational quantum circuits (VQC), Ostaszewski *et al.* [20] and Pirhooshyaran *et al.* [22] attempted to optimize the design parameters of VQC before logic synthesis. Wang *et al.* [32] provided a noise-adaptive search algorithm for VQC by considering the effect of quantum noise. Several studies also focused on optimizing quantum compilation [17, 36], which tried to construct logic quantum circuits based on the specified unitary transformations. Fösel *et al.* [7] optimized the logical quantum circuits by considering the quantum hardware. However, these methods did not address the problem of QCP: an $\mathcal{NP}$-complete problem with large design space and high dimensionality.

Several approaches attempted to adopt ML algorithms for QCP. Paler *et al.* [21] adopted Gaussian Process to estimate the depth of the generated quantum circuits. Although Acampora *et al.* [1] optimized the initial mapping using deep neural networks, their supervised learning method required large numbers of labelled data, and the optimization of SWAP strategy was not addressed in their paper. Pozzi *et al.* [9, 23] attempted to address qubit routing, but the layer-by-layer processing suffered from the sub-suboptimal issue [30]. Also, the initial mapping problem was not addressed in their work. In this paper, we propose an end-to-end ML-based framework for QCP. A policy-based DRL algorithm is used to learn the optimal SWAP strategy. We encode the quantum circuit's information into the state space to improve the optimality. An evolutionary algorithm is then proposed to optimize the initial mapping.

3 PROPOSED METHOD

3.1 Problem Definition & Framework Overview

The problem of QCP can be described as mapping a list of logical quantum gates G into the target quantum device specified by $< P, E >$, where $P = \{p_1, p_2, ..., p_{N_p}\}$ represents physical qubits and $E = \{e_1, e_2, ..., e_{N_e}\}$ denotes N_e edges between adjacent qubits. As the input of QCP is the logically-synthesised quantum circuits, this paper assumes G only contains single-qubit quantum gates G_1 and two-qubit quantum gates G_2. Also, as the number of SWAP gates will only be affected by two-qubit quantum gates [35], we make $G = G_2$ for simplicity. We define the outputs of QCP to be: *i)* an initial mapping $i \in I$, and *ii)* a SWAP strategy $s \in S$. The initial mapping i is a vector of size N_p, which denotes the mapping between physical qubits and logical qubits. The SWAP strategy s is a variable-length vector representing a series of legal SWAP insertion allowed by the underlying quantum devices. In this paper, we formulate the QCP as a bilevel optimization problem as follows:

$$\arg \max_{i \in I, s \in S} \{R_{qcp}(i, s, G, P, E) = R_{init}(i, G, P, E) + R_{swap}(i, s, G, P, E)\} \quad (1)$$

$$\text{s.t.} \quad s \in \arg \max_{t \in S} \{R_{swap}(i, t, G, P, E) = R_{exe}(i, t, G, P, E) - C_{swap}(t)\} \quad (2)$$

Given a quantum circuit with G and i, the lower-level optimization in Equation 2 aims at finding the optimal SWAP strategy s with the minimal number of SWAP insertion $C_{swap}(t)$ and the maximal execution reward $R_{exe}(i, t, G, P, E)$. In the upper-level optimization (Equation 1), the objective function tries to maximize the initial reward $R_{init}(i, G, P, E)$ and objective function $R_{swap}(i, t, G, P, E)$ in the the lower-level problem. To address this bilevel optimization problem, we propose a two-step ML-based framework, which is illustrated in Figure 1. We realize that the lower-level problem (Equation 2) is essentially a combinatorial problem. As RL has recently demonstrated their advantages in combinatorial optimization [14, 15, 37], we adopt a policy-based DRL algorithm to optimize SWAP strategies given a randomly-initialized mapping i. Once the training of our DRL agent is completed, we then optimize the initial reward R_{init} in Equation 1. As this upper-level problem is essentially a discrete search problem with vast design space, we adopt an evolutionary algorithm [16] to minimize the overall SWAP cost, which offers the flexibility of exploring optimality-runtime trade-off.

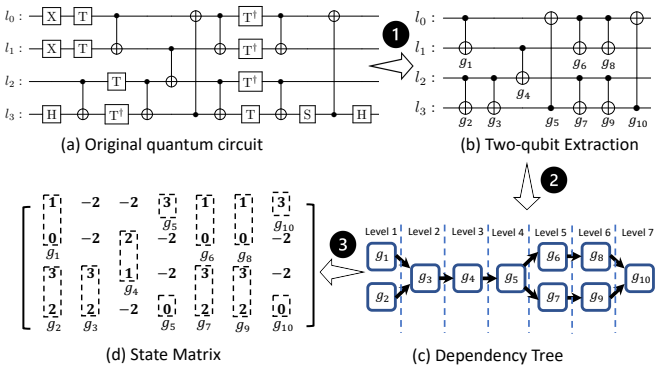

Figure 3: Pre-process to extract state matrix.

3.2 DRL for Swap Strategy Optimization

In the first step of our framework, we train a DRL agent to learn the optimal SWAP strategy. To improve the generalization ability, we randomly generate the initial mappings during the training. This paper defines our DRL agent as follows:

3.2.1 State Space and Pre-Process. As indicated by [29, 30], the layer-by-layer or level-by-level processing may lead to the suboptimal problem. To address this issue, we represent the state using a matrix that contains the whole quantum circuit's information, including the qubit dependencies of all the quantum gates, and the mapping relationship between logical and physical qubits throughout the execution. To encode the information into a matrix, we pre-process the quantum circuit to get the state. There are three operations involved in the pre-processing: *i)* extracting two-qubits gates, *ii)* building dependency tree and *iii)* construction of state matrix, which are illustrated in Figure 3. As the single-qubit gates will not affect the SWAP cost [35], we first extract all the two-qubits gates from the original quantum circuit. Then, a dependency tree is built based on the extracted quantum gates, which indicates the relative execution orders of two-qubits gates. A state matrix is constructed according to the dependency tree and the mapping relationship between logical and physical qubits. We use the i-th row of the state matrix to represent i-th physical qubit and its associated logical qubit. Each column of the matrix state represents a separate time step (level). The element in the i-th row and the j-th column denotes the position of qubit connected to the i-th qubit, which is specified by its corresponding quantum gate at time step j. The value is set to be -2 if no quantum gates are associated with the qubit at a certain time step.

3.2.2 Action Space and Reward. The action space is defined by all the SWAP insertions allowed by the underlying quantum devices. We use an integer ranging from $\{0, 1, ..., N_e\}$ to represent an action, which corresponds to one of edges $e_{1\sim N_e}$ specified by E.

The reward function of our DRL contains four components: *i)* gate reward, which equals to the number of quantum gates executed given the current action, the updated state and hardware constraints, *ii)* done reward, which is applied when all the quantum gates have been executed, *iii)* SWAP penalty when a SWAP gate is inserted, and *iv)* non-execution penalty when there are no executable gates after the SWAP gate is inserted. Note that the gate reward and done reward refer to $R_{exe}(i, t, G, P, E)$ in Equation 2. This paper sets gate reward as +1 per gate, done reward as +10, SWAP penalty as −3, non-execution penalty as +1.

3.2.3 DRL Agent and Optimization Stratety. We define the policy network using a multi-layer perceptron (MLP) with three layers. The hidden sizes are set to be 64, 64 and 96. The policy network receives the state matrix as inputs, and generates an integer after a softmax function to output an action. Another MLP with the same configurations is used as our value network. Proximal Policy Optimization (PPO) [24] is used in our approach to update the parameters of both policy and value networks.

3.2.4 QCP Simulator. We implement a QCP simulator as the environment to interact with our DRL agent. Based on the current action and state matrix, the QCP simulator outputs the accumulated reward and the next state matrix. The pseudocode of the proposed QCP simulator is presented in Algorithm 1. Receiving $Sate_{cur}$ and $Action$ as inputs, the QCP simulator obtains the swapping qubits based on $Action$ and the pre-defined E. Since each row in the state matrix represents a qubit, we update the state matrix by exchanging the rows according to the action taken by the DRL agent. Then, a loop is performed to execute quantum gates level by level, with $Reward$ accumulated during the whole process. The state matrix is updated by left shifting and padding when quantum gates are executed. The loop terminates until there are no executable quantum gates in the current level. The QCP simulator returns the updated state matrix $State_{next}$ and step reward $Reward$ as the output.

3.3 Evolutionary Algorithm

Our adopted EA contains five operations, i.e., population initialization, evaluation, crossover, mutation and selection. They are defined as follows:

- Population initialization: We randomly generate N_{popu} numbers of initial mappings as the original population.
- Evaluation: The trained DRL agent is used to generate SWAP strategies for each initial mapping. The cumulative rewards are then evaluated using the QCP simulator.
- Selection: $Prob_{sel} \times N_{popu}$ mappings with the higher reward are selected as the parent population.
- Mutation: We mutate mappings with a probability $Prob_{mutate}$ using a legal qubit swap provided by the underlying qubit device. The mutation will be skipped if the swap is not allowed.
- Crossover: We combine two mappings from the parent population with a probability $Prob_{cross}$. The crossover will be skipped if the mapping after crossover is invalid.

Algorithm 1 QCP Simulator.

Inputs: $State_{cur}, Action$
Outputs: $State_{next}, Reward$
Row_1, Row_2 = Edge ($Action$) ▷ Get swapped qubit
$Tmp = State_{cur}[Row_1][:]$ ▷ Update state by action
$State_{cur}[Row_1] = State_{cur}[Row_2]$
$State_{cur}[Row_2] = Tmp$
while ($True$) ▷ Update state if executable
  $Column = State_{cur}[:][0]$
  if Executable ($Column$)
    $Reward+ = Reward_{exe}$ ▷ Executable reward
    $State_{cur}[:][:-1] = State_{cur}[:][1:]$ ▷ Shift Left
    $State_{cur}[:][-1] = [-2] * N_{column}$ ▷ Pad using −2
    if Done ($State_{cur}$):
      $Reward+ = Reward_{done}$ ▷ Done reward
      break
  else
    if ($Reward == 0$)
      $Reward- = Penalty_{nonexe}$ ▷ Non-Executable penalty
    break
$State_{next} = State_{cur}$
$Reward- = Penalty_{swap}$ ▷ Swap reward

During the population initialization, we apply several rules to eliminate the invalid initial mapping to speed up the runtime: *i)* The mapped physical qubits are connected, which means there are at least $N_l - 1$ edges existed in the initial mapping. *ii)* All the gates in the first level of the dependency tree are executable under the initial mapping. The objective of EA is defined as the cumulative reward generated from the QCP simulator and the DRL agent.

4 EXPERIMENTS

4.1 Experimental Setup

We implemented our QCP simulator using *OpenAI Gym* toolkit. The RL platform *Tianshou* [33] was used to implement our DRL agent. All the codes were implemented using Python with version 3.7.11. A comprehensive set of benchmarks was collected from [2, 8, 19, 29, 30], which contained 18 quantum circuits with different logical qubits ranging from 3 ∼ 16. We selected three quantum devices: IBM QX2 with 5 qubits, IBM Melbourne with 14 qubits and Rigetti Aspen-4 with 16 qubits. All our evaluations were run on an Ubuntu 16.04 server which had an Intel Xeon E5-2680 v2 CPU (196 GB memory) with 8 CPU cores clocked at 2.4 GHz and a TITAN X Pascal with 3,840 CUDA cores clocked at 1.4 GHz. Note that the GPU was only used for training. For a fair comparison, we evaluated the runtime cost of all different methods using the same CPU. In terms of EA setting, we set $Prob_{sel} = 0.25$, $Prob_{mutate} = 0.5$ and $Prob_{cross} = 0.5$. The population size N_{popu} is 40 for IBM QX2 and $N_{popu} = 800$ for IBM Melbourne and Rigetti Aspen-4.

Table 1: Knowledge Transfer

	Train (Seen) Circuits	**Unseen Circuits**					
Circuit	mod5 mils_65	adder	4mod5 v0_18	4mod5 v0_19	4mod5 v0_20	4mod5 v1_22	mod5 d1_63
Orig SWAP Cost	12	13	36	22	35	19	26
Train Time (GPU)	21.2s	-	-	-	-	-	-
Opt SWAP Cost	(2)	1	5	2	1	1	2
Eval Time (CPU)	-	0.3s	0.7s	0.3s	0.3s	0.3s	0.2s

4.2 Knowledge Transfer

The key advantage of our DRL-based framework is the knowledge transfer: the experience learnt from the previously-seen circuits can be used to optimize the unseen circuits. To demonstrate this, we evaluated six different quantum circuits on IBM QX2 using an untrained DRL agent. The results are treated as the original (Orig) SWAP cost presented in Table 1. Then, we set *mod5mils_65* as the training quantum circuits with the rest as test circuits. As we can see, after training *mod5mils_65* for 21.2 seconds, our framework decreases the SWAP cost significantly on all other unseen circuits. The reduction in the SWAP cost demonstrates the ability of knowledge transfer provided by our framework. In the subsequent experiments, we used *mod_mult_55*, *rc_adder_6* and *barenco_tof_5* as the training circuits for IBM Melbourne and Rigetti Aspen-4.

4.3 Comparison with Heuristic & Exact Methods

As demonstrated in [30], Qiskit [3] and t|ket⟩ [28] are the leading heuristic approaches for QCP. Therefore, we included both Qiskit and t|ket⟩ to demonstrate the advantages of our approach over the heuristic methods. We used t|ket⟩ 0.16 and Qiskit 0.18 for evaluation. While using Qiskit, *SabreLayout* and *SabreSWAP* optimizations [13] were enabled to achieve better results. However, we found these optimizations introduced runtime error on some quantum circuits. On the rest of circuits, we chose the *DenseLayout*, *LookaheadSWAP* and *StochasticSWAP*. For circuits optimized by *StochasticSWAP*, we ran five times to obtain the best results for Qiskit.

Table 2 shows the numbers of SWAP gates required by each quantum circuit under different approaches. For the most of quantum circuits on IBM QX2, Qiskit with *Sabre* optimization performs better than t|ket⟩. However, while targeting on IBM Melbourne and Rigetti Aspend-4, t|ket⟩ costed less number of SWAP gates than Qiskit did on our benchmarks. In comparison with both t|ket⟩ and Qiskit, our ML-based approach outperforms in all our benchmarks on different quantum devices. Compared with Qiskit when IBM QX2 was set as backend quantum device, our method reduced the the number of SWAP gates by 20 on 4*mod*5-*v*0_18. Comparing with t|ket⟩ on IBM Melbourne, our approach decreased the SWAP cost of *vbe_adder_3* by at most 8. For *queko_15_1* on Rigetti Aspend-4, we reduced the SWAP from 38 to 0, which improved the optimality by 100%. Therefore, our ML-based method can achieve higher optimiality on QCP than the leading heuristic approaches.

Table 2: Comparison with Heuristic Methods

Program	N_p	Architecture	No. of SWAP gates		
			Qiskit [3]	t|ket⟩ [28]	Our work (ML-based)
or	3	IBM QX2	0	0	0
adder	4	IBM QX2	1	4	1
qaoa5	5	IBM QX2	0	1	0
mod5mils_65	5	IBM QX2	3	4	2
mod5d1_63	5	IBM QX2	10	6	2
4gt13_92	5	IBM QX2	0	7	0
4mod5-v0_18	5	IBM QX2	25	6	5
4mod5-v0_19	5	IBM QX2	10	2	2
4mod5-v0_20	5	IBM QX2	5	4	1
4mod5-v1_22	5	IBM QX2	3	4	1
barenco_tof_4	7	Melbourne	16	9	5*
tof_4	7	Melbourne	11	1	1
mod_mult_55	9	Melbourne	35	12	9 *
tof_5	9	Melbourne	16	7	1
vbe_adder_3	10	Melbourne	24	16	8*
queko_05_0	16	Aspen-4	24	1	0
queko_10_3	14	Aspen-4	42	15	0
queko_15_1	14	Aspen-4	61	38	0

* Results varied by random seeds and population sizes. The reported one was the best we achieved by running multiple times.

Table 3: Comparison with OLSQ-SWAP

Program	OLSQ-SWAP [29]		Our Work	
	No. of SWAP gates	Time Cost (s) (CPU)	No. of SWAP gates	Time Cost (s) (CPU)
or	0	0.6	0	0.2
adder	1	0.5	1	0.3
qaoa5	0	0.5	0	0.2
mod5mils_65	2	1.3	2	0.3
mod5d1_63	2	0.7	2	0.2
4gt13_92	0	0.3	0	0.3
4mod5-v0_18	5	28.2	5	0.7
4mod5-v0_19	2	1.1	2	0.3
4mod5-v0_20	1	0.2	1	0.3
4mod5-v1_22	1	0.3	1	0.3
barenco_tof_4	5	312.3	5	51.6
tof_4	1	3.6	1	0.3
mod_mult_55	8	1963.8	9	62.8
tof_5	1	6.6	1	0.4
vbe_adder_3	8	2183.6	8	65.7
queko_05_0	0	1.1	0	0.1
queko_10_3	0	15.3	0	0.5
queko_15_1	0	38.2	0	1.7

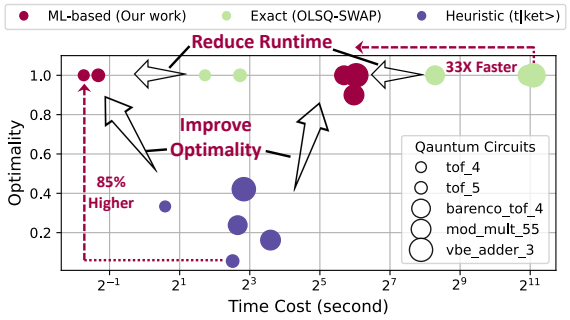

Figure 4: Optimality versus runtime cost on IBM Melbourne.

Among various exact approaches [5, 25, 34, 35], OLQS-SWAP [29] outperforms in terms of the number of SWAP gates and the runtime cost. Therefore, we chose OLQS to represent the leading exact approach in our comparison. In Table 3, we compared our ML-based approach again OLQS in terms of SWAP and runtime costs. Except for *mod_mult_55* where our approach required one more SWAP gate, we achieved the same optimality as the exact search on the rest of the circuits. In terms of runtime cost, our approach achieved up to 40, 33 and 30 times speedup on IBM QX 2, IBM Melbourne and Rigetti Aspen-4 respectively. Therefore, our work can achieve the same level of SWAP cost as the best exact approach does while significantly reducing the runtime cost.

To visualize the advantages of our framework over both heuristic and exact methods, Figure 4 presents the runtime cost and optimality of five different quantum circuits on IBM Melbourne using different approaches. We measured the optimality by dividing the required SWAP cost from the optimal SWAP cost. Our ML-based

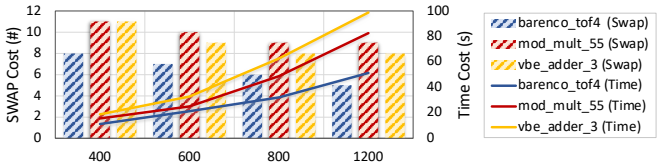

Figure 5: Swap and time cost under different EA settings.

framework achieved 85% higher optimality than t|ket⟩ on tof_5. Compared with OLSQ-SWAP, we achieved the same optimality while decreasing the runtime cost by 33× on vbe_adder_3.

4.4 Flexibility of Exploring Trade-off

While achieving the higher optimality than heuristic methods and the shorter runtime than the exact approaches, our ML-based framework also enables users to explore optimality-runtime trade-off. To demonstrate this, we evaluated three quantum circuits on IBM Melbourne, i.e., $barenco\_tof\_4$, mod_mult_55 and vbe_adder_3. We set different population sizes while performing the EA optimization: 400, 600, 800 and 1200. Figure 5 presents the SWAP and time costs of three different quantum circuits under different population sizes. A decreasing trend was shown in the SWAP cost when the population size and runtime cost increased. It can also be observed that the optimality and runtime trade-off varies circuit by circuit. For instance, $barenco\_tof\_4$ and vbe_adder_3 achieved the lowest SWAP cost when the population size was 1200 and 800 respectively. By defining these customizable parameters, users are able to explore the trade-off on their own quantum circuits and devices.

5 CONCLUSION

This work proposes a novel machine learning (ML)-based framework for quantum circuit placement (QCP). We adopt a policy-based deep reinforcement learning algorithm to optimize the mapping strategy. Then, an evolutionary algorithm is proposed to optimize the initial mapping. Compared with the leading heuristic industrial approaches, our ML-based method is able to achieve up to 100% higher optimality. In comparison with exact search, we reduce the runtime by up to 40 times while keeping the same level of optimality. In future, we aim to evaluate our method on large-scale quantum circuits, consider quantum fidelity into our framework and explore noise-adaptive method for QCP.

ACKNOWLEDGEMENT

The support of UK EPSRC grants (UK EPSRC grants EP/L016796/1, EP/N031768/1, EP/P010040/1, EP/V028251/1 and EP/S030069/1) is gratefully acknowledged.

REFERENCES

[1] Giovanni Acampora and Roberto Schiattarella. 2021. Deep neural networks for quantum circuit mapping. *Neural Computing and Applications* (2021), 1–21.
[2] Matthew Amy et al. 2013. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 6 (2013), 818–830.
[3] MD SAJID ANIS et al. 2021. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2573505
[4] Alán Aspuru-Guzik et al. 2005. Simulated quantum computation of molecular energies. *Science* 309, 5741 (2005), 1704–1707.
[5] Debjyoti Bhattacharjee et al. 2019. MUQUT: Multi-constraint quantum circuit mapping on NISQ computers. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–7.
[6] Andrew M Childs et al. 2019. Circuit transformations for quantum architectures. *arXiv preprint arXiv:1902.09102* (2019).
[7] Thomas Fösel et al. 2021. Quantum circuit optimization with deep reinforcement learning. *arXiv preprint arXiv:2103.07585* (2021).
[8] Pranav Gokhale et al. 2020. Optimized quantum compilation for near-term algorithms with openpulse. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 186–200.
[9] Steven Herbert and Akash Sengupta. 2018. Using reinforcement learning to find efficient qubit routing policies for deployment in near-term quantum computers. *arXiv preprint arXiv:1812.11619* (2018).
[10] J Hsu. 2018. Intels 49-Qubit Chip Shoots for Quantum Supremacy.
[11] Julian Kelly. 2018. A preview of Bristlecone, Google's new quantum processor. *Google Research Blog* 5 (2018).
[12] Will Knight. 2017. IBM raises the bar with a 50-qubit quantum computer. *Sighted at MIT Review Technology* (2017).
[13] Gushu Li et al. 2019. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1001–1014.
[14] Nina Mazyavkina et al. 2021. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research* (2021), 105400.
[15] Azalia Mirhoseini et al. 2020. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746* (2020).
[16] Melanie Mitchell. 1998. *An introduction to genetic algorithms*. MIT press.
[17] Lorenzo Moro et al. 2021. Quantum Compiling by Deep Reinforcement Learning. *arXiv preprint arXiv:2105.15048* (2021).
[18] Prakash Murali et al. 2019. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 527–540.
[19] Yunseong Nam et al. 2018. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information* 4, 1 (2018), 1–12.
[20] Mateusz Ostaszewski et al. 2021. Reinforcement learning for optimization of variational quantum circuit architectures. *arXiv preprint arXiv:2103.16089* (2021).
[21] Alexandru Paler et al. 2020. Machine learning optimization of quantum circuit layouts. *arXiv preprint arXiv:2007.14608* (2020).
[22] Mohammad Pirhooshyaran and Tamas Terlaky. 2020. Quantum Circuit Design Search. *arXiv preprint arXiv:2012.04046* (2020).
[23] Matteo G Pozzi, Steven J Herbert, Akash Sengupta, and Robert D Mullins. 2020. Using reinforcement learning to perform qubit routing in quantum compilers. *arXiv preprint arXiv:2007.15957* (2020).
[24] John Schulman et al. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[25] Alireza Shafaei et al. 2013. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[26] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.
[27] Marcos Yukio Siraichi et al. 2018. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization (CGO)*. 113–125.
[28] Seyon Sivarajah et al. 2020. t|ket>: A retargetable compiler for NISQ devices. *Quantum Science and Technology* 6, 1 (2020), 014003.
[29] Bochen Tan and Jason Cong. 2020. Optimal layout synthesis for quantum computing. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.
[30] Bochen Tan and Jason Cong. 2020. Optimality study of existing quantum computing layout synthesis tools. *IEEE Transactions on Computers (TC)* (2020).
[31] Swamit S Tannu and Moinuddin K Qureshi. 2019. Not all qubits are created equal: a case for variability-aware policies for NISQ-era quantum computers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 987–999.
[32] Hanrui Wang et al. 2021. Quantumnas: Noise-adaptive search for robust quantum circuits. *arXiv preprint arXiv:2107.10845* (2021).
[33] Jiayi Weng et al. 2021. Tianshou: A Highly Modularized Deep Reinforcement Learning Library. *arXiv preprint arXiv:2107.14171* (2021).
[34] Robert Wille et al. 2014. Optimal SWAP gate insertion for nearest neighbor quantum circuits. In *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 489–494.
[35] Robert Wille et al. 2019. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[36] Yuan-Hang Zhang et al. 2020. Topological quantum compiling with reinforcement learning. *Physical Review Letters* 125, 17 (2020), 170501.
[37] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).
[38] Alwin Zulehner et al. 2018. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 38, 7 (2018), 1226–1236.