Invited: Leveraging Machine Learning for Quantum Compilation Optimization

Xiangyu Ren
University of Edinburgh
Edinburgh, United Kingdom
Tencent Quantum Lab
Shenzhen, Guangdong, China
xiangyu.ren@ed.ac.uk

Tianyu Zhang
Tencent Quantum Lab
Shenzhen, Guangdong, China
miketyzhang@tencent.com

Xiong Xu
Tencent Quantum Lab
Shenzhen, Guangdong, China
walterxu@tencent.com

Yi-Cong Zheng
Tencent Quantum Lab
Shenzhen, Guangdong, China
yicongzheng@tencent.com

Shengyu Zhang
Tencent Quantum Lab
Shenzhen, Guangdong, China
shengyzhang@tencent.com

Abstract

The design of quantum algorithms typically assumes the availability of an ideal quantum computer, characterized by full connectivity, noiseless operation, and unlimited coherence time. However, Noisy Intermediate-Scale Quantum (NISQ) devices present a stark contrast, with a limited number of qubits, non-negligible quantum operation errors, and stringent constraints on the connectivity of physical qubits within a Quantum Processing Unit (QPU). This necessitates the dynamic remapping of logical qubits to physical qubits within the compiler to facilitate the execution of two-qubit gates in the algorithm. However, this introduces additional operations, consequently reducing the fidelity of the algorithm. Therefore, minimizing the number of added gates becomes crucial. Finding such an optimal routing problem is NP-hard, and the task is conventionally addressed using human-crafted heuristics to search for SWAP sequences, but these lack performance guarantees. In this study, we employ a Seq2Seq machine learning model for the qubit routing task, incorporating a Transformer neural network to learn the routing information in the gate and SWAP sequence. Compared to heuristic search-based algorithms, our approach significantly reduces the overhead of quantum computing resources required to adapt logical circuits to physical circuits executable on specific quantum backend hardware.

Keywords

Quantum Computation, Compiler Optimization, Machine Learning, Qubit Routing

ACM Reference Format:
Xiangyu Ren, Tianyu Zhang, Xiong Xu, Yi-Cong Zheng, and Shengyu Zhang. 2024. Invited: Leveraging Machine Learning for Quantum Compilation Optimization. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3649329.3663510

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3663510

1 Introduction

Quantum computing (QC) possesses the capability to address many problems that are intractable for classical computers [10]. Recent advancements have seen significant improvements in the development of quantum computers across various physical platforms, including superconducting qubits, trapped ions, and neutral-atom arrays. However, the performance of current Noisy Intermediate-Scale Quantum (NISQ) devices [11] is limited by their imperfect operations, which prevent them from achieving their full computational potential.

In this context, a quantum compiler plays a crucial role by converting high-level quantum algorithms into executable instructions on quantum hardware. This process involves qubit mapping, gate decomposition, and the insertion of SWAP gates to meet the connectivity constraints of quantum architectures [8]. Regrettably, these necessary adaptations increase the complexity of operations, exacerbate the effects of noise, and diminish overall efficiency. Optimizing the compilation process is therefore essential to reduce overhead costs and improve the robustness and efficiency of quantum computations.

Our research primarily focuses on optimizing qubit routing to minimize the number of required SWAP gates. While asymptotically optimal methods [19, 20] exist, their performance on individual instances may deviate by a constant factor from the optimal solution. Exact solvers [9, 14] can yield optimal results, but their practical application is hindered by scalability issues. Moreover, efficient heuristic methods such as the bidirectional heuristic search [5] and filtered depth-limited search [6] often fall short of reaching optimal solutions. Reinforcement learning-based compilers have shown promise on smaller circuits, but their scalability and effectiveness on larger circuits remain to be validated [4].

To address these limitations, we propose a Sequence-to-Sequence (Seq2Seq) machine learning model to enhance qubit routing within our quantum compilation framework [13]. This model employs a Transformer neural network architecture [16] to process gate sequences and generate the required SWAP sequences effectively.

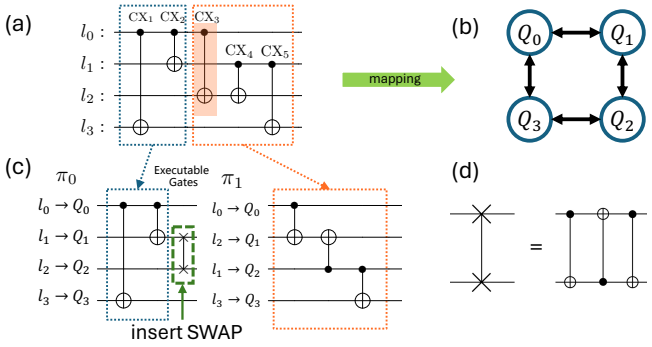

Figure 1: An example of the qubit routing process. (a) Original logical circuit of 5 CXs. (b) Target QPU backend. (c) The compiled circuit on QPU. (d) SWAP gate decomposition.

As detailed in Section 4, our approach exhibits significant advancements compared to state-of-the-art heuristic compilers [5, 6], achieving an average reduction of 71% and 30% in the number of inserted SWAP gates, respectively. Additionally, it accelerates the compiler runtime by an average of 138x on IBM Q20 Tokyo.

2 Background and Motivation

2.1 Qubit Mapping and Routing

We begin with a practical example to illustrate the problem of qubit routing, as depicted in Fig. 1. A quantum circuit composed of 5 CXs (Fig. 1(a)) is to be executed on a 4-qubit QPU backend (Fig. 1(b)). The initial mapping is π_0 as shown on the left of Fig. 1(c), for which CX_1 and CX_2 are directly executable. However, CX_3 cannot be executed since Q_0 and Q_2 are not directly coupled. A standard solution is to swap the states of Q_1 and Q_2, which, in this case, enables the execution of CX_3, CX_4 and CX_5. The inserted SWAP gate updates the qubit mapping status to π_1 (Fig. 1(c)), and it can be implemented by three CX gates (Fig. 1(d)). From this example, it is also evident that single-qubit gates do not impact qubit mapping and routing. Therefore, we will focus solely on circuits containing two-qubit gates.

In compiling deep quantum circuits with a substantially larger number of qubits, finding a qubit routing solution that minimizes the number of SWAP insertions significantly impacts the performance of circuit execution. That is because an excessive number of extra SWAPs significantly expand the circuit depth and size, aggravating the influence of qubit noise. Several heuristic approaches have been proposed to tackle this problem. Zulehner *et al.* [21] employ an exhaustive searching approach to traverse all the possible SWAP combinations. Li *et al.* [5] proposed the SWAP-based bidirectional heuristic search algorithm, finding the SWAP operation by minimizing the distance of qubit pairs in the front layer of the circuit. In Ref. [6], the authors define search depth k as considering all possible SWAP sequences up to length k. The authors enhance the fixed-depth heuristic search algorithm with filtering and fallback operation. One can observe that increasing k in the heuristic method can give significant rise to the optimality of its solution. However, it leads to an exponential rise in the search cost. Figure 2 shows how the number of added SWAPs, normalized to the compiler's output in this work, varies with search time cost for different routing algorithms on IBM Q20.

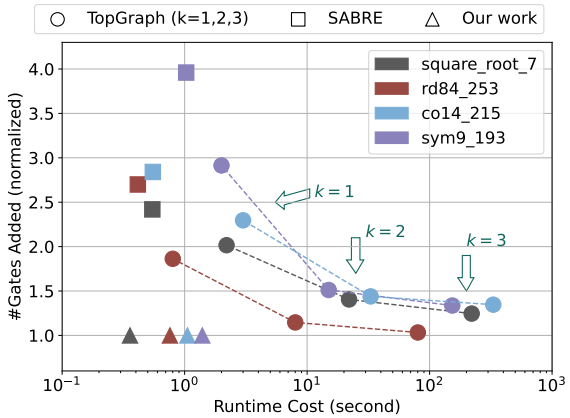

Figure 2: The normalized number of inserted SWAP gates (#Gates) and the runtime cost are compared for three compilers: TopGraph [6](circle), SABRE [5](square), and our method (triangle), across four quantum circuits on IBM Q20. It can be observed that SABRE [5] (square) generally has a small time cost, and TopGraph [6] (circle) has a small #Gates cost given enough runtime. In comparison, our method significantly reduces the #Gates to even better than that in TopGraph, while keeping a runtime similar to that in SABRE.

3 ML-based Optimization

We develop a machine learning (ML)-based optimization method, leveraging the Transformer neural network [16] model to learn the SWAP insertion strategy for qubit routing. The operational framework of our model takes sequences of qubit gates as inputs and produces the necessary SWAP operations as outputs, considering the specific connection topology of QPU. Training datasets are collected independently for various backend connection topologies.

The framework design is shown in the green box of Fig. 3. In this work, we train Transformer models separately for distinct backend connection topologies to generate the SWAP sequence needed for routing. To ensure that the generated SWAP sequence effectively aids the qubit routing process, it is evaluated using an evaluation function to assess its quality. Specifically, the heuristic evaluation function in our framework is defined as follows:

$$eval(\pi) = (\#g_{exe} \times \beta + \frac{1}{k}\sum_{i=0}^{L} \alpha^{lyr(g[i])} \times (\gamma - dist(g[i], \pi)). \quad (1)$$

Here, π represents the qubit mapping, $\#g_{exe}$ denotes the number of executable two-qubit gates in the input gate sequence given the qubit mapping π, and g represents the next L input gates in the circuits. Additionally, $lyr(g[i])$ refers to the index of the gate layer containing $g[i]$ in the input circuit, with the layers indexed starting from the front gate layer. The variable $dist$ represents the topology distance between two physical qubits supporting the two-qubit gate, as described in Ref. [5]. Furthermore, we introduce adjustable parameters α, β, and γ.

3.1 Algorithm Design

In this section, we show how to train and apply Transformer [2, 17], an ML model originally designed for natural language processing tasks, for quantum compilation. Given a sequence of logical quantum gates C_ℓ, we first execute the gates that are compatible with

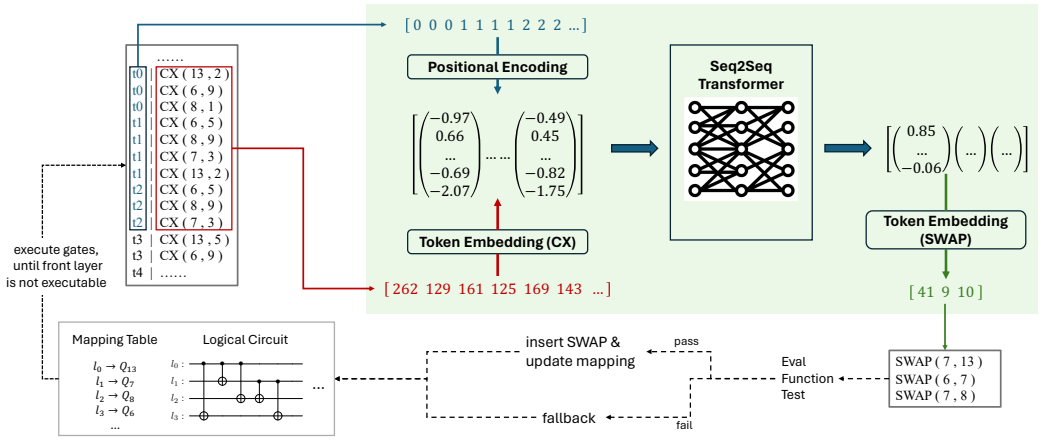

Figure 3: An overview of our compilation framework.

qubit connectivity on hardware until we encounter a gate that is not directly executable. We then feed the first L gates in C_ℓ to the trained Transformer model to generate a SWAP sequence s which ideally aids the rest execution. We use the *eval* function to determine whether inserting s provides any gain. If it does, we accept it and proceed. If not, we employ a "fallback" heuristics [6] to resolve the first non-executable gate. Details of the Transformer-based routing procedure are given in Algorithm. 1 and Figure 3.

Algorithm 1 ML-optimized qubit routing

Input: Initial mapping π_0, logical quantum circuit C_ℓ, Seq2Seq neural network model $NNmodel$, evaluation function $eval$, number L of lookahead gates.
Output: Compiled quantum circuit C_p with physical qubit index.

$\pi \leftarrow \pi_0, C_p \leftarrow \emptyset$
while $C_l \neq \emptyset$ **do**
  $s_e \leftarrow$ executable gate sequence prefix under (C_ℓ,π)
  $C_\ell \leftarrow C_\ell - s_e, C_p \leftarrow C_p + s_e$
  $s_f \leftarrow$ the gate sequence of the first L gates in C_ℓ
  Output SWAP sequence $s \leftarrow NNmodel(s_f)$
  $\pi' \leftarrow$ update π using s
  if $eval(\pi') > eval(\pi)$ **then**
    $\pi \leftarrow \pi', C_p \leftarrow C_p + s$
  else
    generate fallback SWAP gate sequence s'
    $\pi \leftarrow$ update π using $s', C_p \leftarrow C_p + s'$
  end if
end while

Token Embedding. In our compiler, we begin by converting gate information from the logical circuit into tensors, which are then fed as input to the Transformer. This process involves combining the gate information with the current qubit mapping status. Initially, we translate the gates from logical qubit indexing to physical qubit indexing. Subsequently, each physical qubit gate is represented by a token number (depicted as red numbers in Fig. 3), based on the indices of the physical qubits involved. Finally, a token embedding layer is employed to convert these tokens into tensors with dimension 1024, utilizing the token embedding model provided by PyTorch.

Positional Encoding. In the NLP task, position information is included with each token in the input sequence to help the Transformer model understand the structure of the sequence. In this study, the model processes the first $L = 10$ gates from the original circuit as the input sequence. The position of each two-qubit gate within this sequence is indicated by its layer index in the logical circuit, starting from the front layer (depicted as blue numbers in Fig. 3). We employ the positional encoding algorithm described in Ref. [16] to facilitate this encoding.

Sequence Padding. The output length of the Transformer model in this work is fixed at $k = 3$ (the green numbers in Fig. 3). In case the output SWAP sequence depth is less than k, we pad 0 at the end to make the length equal to k.

3.2 Training Stage

Given a QPU backend topology, the learning process is supervised. It involves randomly generated circuits with extended sequences of two-qubit gates applied to two arbitrary physical qubits. We use a heuristic approach to search for the best initial mapping [1], by maximizing an *eval* function in Eq. (1). Combined with the generated initial mapping set Π, these random circuits form the input set $\mathcal{I}$ for the training data. For each sequence in $\mathcal{I}$, we leverage a novel fix-depth filtered searching (Ref. [1]) to obtain a high-quality solution of SWAP sequence, which is used as the output label O of the training data.

4 Evaluation

4.1 Experiment Setup

Our compilation framework is implemented in C++, with model training conducted in Python using PyTorch. The Transformer model is built with PyTorch.nn. Evaluations were performed on a CentOS-7 server featuring an Intel Xeon Platinum 8255C CPU (320 GB memory) with 80 CPU cores at 2.5 GHz, and a NVIDIA V100 GPU with 5120 CUDA cores at 1.2 GHz. We collected 640K data points for each QPU backend, and the entire training process, including dataset generation, took 6 hours.

4.2 Benchmarks and QPU backends

We selected our benchmarks from [6] and [21], which comprise 23 quantum circuits with logical qubits ranging from 5 to 16. Our compilation targets various quantum chip topologies, including the

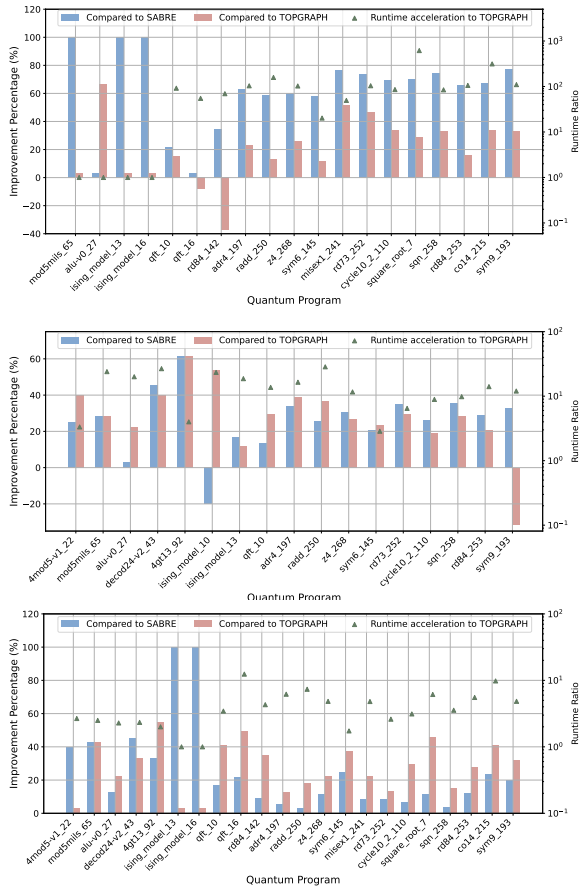

Figure 4: Improvement in SWAP insertion percentages compared to TopGraph and SABRE, and runtime acceleration against TopGraph, for target QPU backends: (a) IBM Q20 Tokyo, (b) Surface-13, and (c) IBM Falcon.

20-qubit IBM Q20 Tokyo, the 27-qubit IBM Falcon featuring a heavy-hexagon topology [3], and the 13-qubit Surface-13 topology [15].

4.3 Results

We compare our work with the state-of-art heuristic algorithms SABRE [5] (latest version is accessed by using *SabreLayout* and *SabreSWAP* in Qiksit [12]) and TopGraph [6]. Fig. 4 shows the improvement in terms of both the number of added gates and compilation runtime towards each quantum circuit. On average across these benchmarks, the number of added gates is reduced by 71%, 31%, and 26% compared to [5], and by 30%, 28%, 27% compared to [6] on IBM Q20 Tokyo, Surface-13 and IBM Falcon, respectively.

We also note that the improvement in compilation runtime over TopGraph [6] varies across different QPU backend topologies. For IBM Q20 Tokyo, the average runtime enhancement is 138x, compared to 14x for the Surface-13 backend and 5x for the IBM Falcon backend. This discrepancy arises from variations in the number of couplings within the QPU backends' topologies.

The IBM Q20 boasts more couplings per qubit, providing a broader range of SWAP sequence options. Consequently, fixed-depth search methods face exponentially larger search spaces. In contrast, the compilation runtime of our Seq2Seq model increases linearly, making it more scalable for larger quantum computations.

5 Conclusion

In this study, we tackle the qubit routing problem employing a Seq2Seq machine learning model with a Transformer architecture, which significantly enhances the efficiency and reduces the number of added gates in quantum circuits. Our findings highlight the considerable potential of integrating machine learning with quantum computing to optimize computational performance [7, 18].

References

[1] Manuscript under preparation.

[2] Pytorch transformer model document. https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html.

[3] Christopher Chamberland, Guanyu Zhu, Theodore J Yoder, Jared B Hertzberg, and Andrew W Cross. Topological and subsystem codes on low-degree graphs with flag qubits. Physical Review X, 10(1):011022, 2020.

[4] Hongxiang Fan, Ce Guo, and Wayne Luk. Optimizing quantum circuit placement via machine learning. In Proceedings of the 59th ACM/IEEE Design Automation Conference, DAC '22, page 19–24, New York, NY, USA, 2022. Association for Computing Machinery.

[5] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '19, page 1001–1014, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Sanjiang Li, Xiangzhen Zhou, and Yuan Feng. Qubit mapping based on subgraph isomorphism and filtered depth-limited search. IEEE Transactions on Computers, 70(11):1777–1788, 2021.

[7] Zhiding Liang, Gang Liu, Zheyuan Liu, Jinglei Cheng, Tianyi Hao, Kecheng Liu, Hang Ren, Zhixin Song, Ji Liu, Fanny Ye, and Yiyu Shi. Graph learning for parameter prediction of quantum approximate optimization algorithm, 2024.

[8] Daniel A Lidar and Todd A Brun. Quantum error correction. Cambridge University Press, 2013.

[9] Wan-Hsuan Lin, Jason Kimko, Bochen Tan, Nikolaj Bjørner, and Jason Cong. Scalable optimal layout synthesis for nisq quantum processors. In 2023 60th ACM/IEEE Design Automation Conference (DAC), pages 1–6, 2023.

[10] Michael A Nielsen and Isaac L Chuang. Quantum computation and quantum information. Cambridge University Press, 2010.

[11] John Preskill. Quantum computing in the nisq era and beyond. Quantum, 2:79, 2018.

[12] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[14] Bochen Tan and Jason Cong. Optimal layout synthesis for quantum computing. In Proceedings of the 39th International Conference on Computer-Aided Design, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery.

[15] Yu Tomita and Krysta M Svore. Low-distance surface codes under realistic quantum noise. Physical Review A, 90(6):062320, 2014.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[18] Hanrui Wang, Zhiding Liang, Jiaqi Gu, Zirui Li, Yongshan Ding, Weiwen Jiang, Yiyu Shi, David Z. Pan, Frederic T. Chong, and Song Han. Torchquantum case study for robust quantum circuits. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery.

[19] Pei Yuan, Jonathan Allcock, and Shengyu Zhang. Does qubit connectivity impact quantum circuit complexity? IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 43(2):520–533, 2024.

[20] Pei Yuan and Shengyu Zhang. Full characterization of the depth overhead for quantum circuit compilation with arbitrary qubit connectivity constraint. arXiv preprint arXiv:2402.02403, 2024.

[21] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the ibm qx architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 38(7):1226–1236, 2019.