





Received 17 April 2023; revised 11 July 2023; accepted 21 July 2023; date of publication 4 August 2023;  
date of current version 14 September 2023.

Digital Object Identifier 10.1109/TQE.2023.3301899

# Machine-Learning-Based Qubit Allocation for Error Reduction in Quantum Circuits

TRAVIS LECOMPTE<sup>1</sup> , FANG QI<sup>2</sup>, XU YUAN<sup>3</sup>  (Senior Member, IEEE),  
NIAN-FENG TZENG<sup>4</sup> (Life Fellow, IEEE),  
M. HASSAN NAJAFI<sup>4</sup>  (Member, IEEE),  
AND LU PENG<sup>2</sup>  (Senior Member, IEEE)

<sup>1</sup>Louisiana State University, Baton Rouge, LA 70803 USA

<sup>2</sup>Tulane University, New Orleans, LA 70118 USA

<sup>3</sup>University of Delaware, Newark, DE 19716 USA

<sup>4</sup>University of Louisiana at Lafayette, Lafayette, LA 70503 USA

Corresponding author: Lu Peng (e-mail: lpeng3@tulane.edu).

This work was supported by the National Science Foundation under Grant OIA-2019511. The work of Travis LeCompte was supported by a Louisiana Board of Regents Graduate Fellowship.

**ABSTRACT** Quantum computing is a quickly growing field with great potential for future technology. Quantum computers in the current noisy intermediate-scale quantum (NISQ) era face two major limitations: 1) qubit count and 2) error vulnerability. Although quantum error correction methods exist, they are not applicable to the current size of computers, requiring thousands of qubits, while current NISQ systems have hundreds at most. It is, therefore, imperative to improve the reliability of the circuits as much as possible to make them robust to the errors that will occur. One common approach is to adjust the compilation process of a circuit to create a final circuit with improved reliability. However, there are many decisions to be made when compiling that affect the final performance of the circuit, two of the most critical ones being the mapping of logical to physical qubits (the qubit allocation problem) and the movement of qubits to satisfy two-qubit gate adjacency requirements (the qubit routing problem). We focus on solving the qubit allocation problem and identifying initial layouts that reduce error. To identify these layouts, we combine reinforcement learning with a graph neural network (GNN)-based Q-network for analyzing both the connections and error rates of the graphlike backend of superconducting quantum computers to make mapping decisions, creating a GNN-assisted compilation (GNAQC) strategy. We provide both the circuit and the properties of the target backend as input to guide the decision-making process. We work with the IBM Qiskit applications programming interface to compile and simulate our quantum circuits. We train the architecture using a set of four backends and six circuits and find that GNAQC generally outperforms preexisting qubit allocation algorithms, increasing final relative output fidelity by roughly 12.7%.

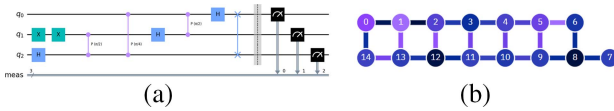
**INDEX TERMS** Fidelity, graph neural networks (GNNs), quantum compilation, qubit allocation.

## I. INTRODUCTION

Quantum computing has quickly become a popular field of research with great potential for future technology. Taking advantage of quantum mechanics allows for several possible operations and interactions that are not possible with classical systems. Quantum systems have the potential to improve communication, encryption, physical simulation, and some algorithms such as Shor's algorithm for factorization [40]. There are several potential physical implementations of quantum information systems, although the two

most available systems today are utilizing superconducting technology [11] and trapped ions [24]. We design and verify our compilation methodology with superconducting quantum computers as they are accessible and have considerable software support. However, the technologies still face limitations, mainly in size and reliability.

Modern quantum computers are classified as noisy intermediate-scale quantum (NISQ) devices [1]. These NISQ devices are named as such due to their limitations on both the number of qubits (or quantum bits) available and the



**FIGURE 1.** (a) Example quantum circuit, displaying a three-qubit QFT algorithm. (b) Example IBM backend. Darker colors denote higher gate error rates (1e-2) while lighter colors denote lower error rates (1e-3).

reliability of these qubits and their operations. Most NISQ devices contain from 10 to 100 noisy qubits, although many systems are on the smaller end of this range, containing only 5–24 qubits. A common metric to evaluate these systems is quantum volume [12], which incorporates both the number of qubits and their vulnerability to error. Due to the relatively high error rates in quantum computers, many executions of algorithms are unlikely to complete without some error. As such, much effort has been put forth to both make the algorithms resilient and to reduce the vulnerability of the physical qubits.

Quantum error correction (QEC) methods do exist but many are not feasible on NISQ systems [5], [9], [16], [21], [43]. Many QEC methods implement similar mechanics to classical replication or redundancy systems, where the data in one or more bits are encoded into a larger number of bits to reduce the effect of incident errors. However, due to the quantum no-cloning theorem [49], rather than copying bits, one must rely on entanglement instead. Again, one or more qubits can be entangled with a greater number of qubits to provide redundancy and mitigate the effect of errors. Shor’s code, the first to demonstrate the existence of QEC methods, encode one qubit into nine to account for both phase and magnitude errors [41]. However, when qubits are a limited resource, it is not possible to implement these QEC methods while retaining enough qubits for computation. Many systems may not be large enough to allow for even one secure qubit depending on the error codes used.

Most approaches increase the reliability of quantum circuits during execution rather than completely removing errors. It is common to modify the circuit during compilation to choose more reliable configurations when applying the circuit to a physical backend. Using different qubits, connections, and operations can have a large impact on the outcome of the circuit as the qubits may exhibit very different error profiles, as demonstrated in Fig. 1, where the color of qubits and their connections indicate their error rates. These error rates can vary day-to-day with the environmental conditions, as we will demonstrate in Section III. The problem is generally broken up into two parts: 1) choosing the initial layout to map virtual qubits of the circuit to physical qubits of the backend (qubit allocation, qubit mapping, layout selection) and 2) moving qubits through the mesh using swap operations to satisfy adjacency requirements for two-qubit operations (qubit routing, SWAP mapping). Due to the large number of possibilities when applying a circuit to a backend, it is difficult to identify the best possible configuration, although many works have found success with a variety of methods [15], [29], [34], [45].

Our work aims to improve upon existing qubit allocation approaches, as our investigation shows that there are considerable performance improvements to be made. To solve the qubit allocation problem, we incorporate graph neural networks (GNNs) to aid in processing the inherent graph representation of the superconducting quantum backend, creating a GNN-assisted compilation strategy (GNAQC). We combine this GNN processing of the backend with feed-forward networks for processing input circuits to create a total system for providing suggested layouts as solutions to the qubit allocation problem. We implement GNAQC using Qiskit [38] and TensorFlow [2] and evaluate its performance on two different IBM backend configurations and six different quantum circuits. We find that GNAQC generally outperforms the other layout methods with some variation across the backends and circuits, increasing relative fidelity by approximately 12.7%. We also find that GNAQC is more consistent at choosing more effective layouts, providing a more reliable allocation method.

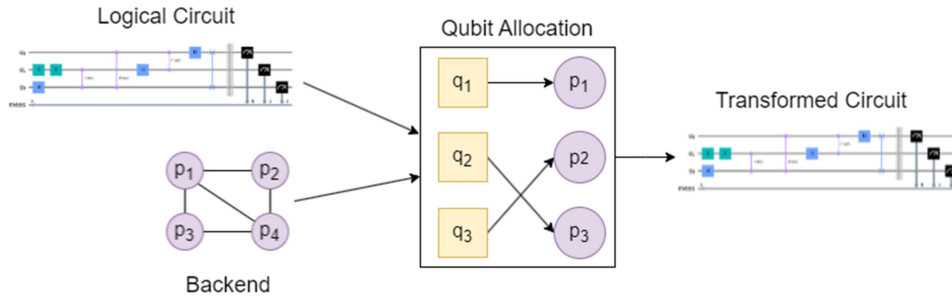
Our contributions can be summarized as follows.

- We demonstrate the limitations of preexisting layout methods.
- We provide GNAQC, a new solution to the qubit allocation problem built on GNNs with feedforward networks.
- We test GNAQC on two physical backends of 7 and 27 qubits using 6 different benchmarks and find that GNAQC can consistently provide better or comparable initial layouts to preexisting methods.
- We demonstrate that GNAQC reduces the error of quantum circuits by providing more reliable layouts, finding a 12.7% relative increase in fidelity.

## II. BACKGROUND

### A. QUANTUM CIRCUITS AND COMPILATION

Quantum circuits are composed of a set of virtual qubits and lists of gates that operate on these qubits. An example circuit is shown in Fig. 1(a). Gates are normally classified into three types: 1) single-qubit; 2) two-qubit; and 3) measurement gates. Single qubit gates simply adjust the state of the qubit it operates on. Two-qubit gates perform operations on a pair of qubits; for example, controlled two-qubit gates modify a target qubit based on the state of a control qubit. Measurement gates measure the qubit and output the result into a classical bit. It is important to note that superconducting systems cannot currently support any-to-any connectivity between qubits, where two-qubit operations can be performed arbitrarily between any two physical qubits. Two-qubit operations can only be performed between pairs of adjacent qubits as shown in Fig. 1(b). If the circuit requires that an operation be performed between two distant qubits, the qubits must be moved from qubit to qubit using pairwise SWAP operations until they reside on adjacent physical qubits. This process of adding SWAP operations to the circuit to satisfy two-qubit operation requirements is known as qubit routing and is another actively researched area in quantum computing.



**FIGURE 2.** Overview of qubit allocation, one step of the compilation process that maps a logical circuit onto a physical quantum machine.

Although many different single- and two-qubit operations, or even multiqubit operations involving three or more qubits, may be performed throughout an algorithm, these operations are normally decomposed and represented as a combination of basis gates. Each backend contains a set of basis gates that act as a universal set from which all other operations can be composed. For example, a SWAP operation could be implemented as three CNOT gates. It is common to see a combination of single-qubit rotation operations and one two-qubit operation as the basis set, although the set of available gates can vary from backend to backend, especially if they are implemented with different technologies. This process of decomposing circuits' gates into basis gates is another major step in compiling a quantum circuit to prepare it for execution on a quantum machine. The general data flow for compiling a quantum circuit is shown in Fig. 2. We omit decomposition from the figure and show the three other steps that are commonly discussed: 1) optimization, 2) allocation, and 3) routing. All of these processes are implemented similarly to compilation passes of classical programs, where the program is gradually modified as it moves from stage to stage of the compilation process. Both allocation and routing are critical to not only the operation of the circuit but also its vulnerability to error, as the error rates of qubits and connections between qubits can vary greatly across the computer.

The most relevant compilation step to this work is qubit allocation or selecting the initial positions to map virtual qubits of the circuit to physical qubits of the backend. Although the allocation method generally does not have as much of an impact on the performance of the circuit as the routing method, the initial position of qubits can impact the choices presented to the routing method and can thus have a substantial effect, particularly for shorter circuits. Qubit allocation methods are normally chosen either to minimize the depth of the circuit (the length of the critical path of operations) or to maximize the success rate of the circuit. This is normally done by minimizing the number of necessary SWAPs inserted by the associated routing method, placing virtual qubits on the most reliable physical qubits, placing virtual qubits in the most highly connected regions of the mesh, or a combination of these methods.

Optimization of the circuit normally occurs in two stages, both before and after the circuit is mapped to the backend hardware through the allocation and routing meth-

ods. Thus, the circuit is optimized at both the virtual level before application and the resulting circuit after application. This allows for high-level optimizations to be performed while the circuit is still in a more conceptual stage, and for lower-level optimizations afterward when the circuit has been transformed. Typically, these optimizations involve removing unnecessary or redundant operations (such as operations that cancel each other out), combining two or more operations into a single operation that produces the same result, or preparing states using additional unused qubits for future operations. These optimizations can shorten the depth of the circuit and, therefore, increase the resilience of the circuit by having fewer operations and thus fewer chances for errors to occur.

## B. MEASUREMENT AND ERROR

Unlike in classical systems, it can be difficult to obtain consistently correct outcomes even without error. Quantum circuits commonly complete execution by being measured back into classical bits. The measurement collapses a state in superposition into one of the two classical bits with probabilities based on their coefficients. Specifically, measuring an unknown state  $\alpha|0\rangle + \beta|1\rangle$  gives state 0 with probability  $|\alpha|^2$  and state 1 with probability  $|\beta|^2$ . This holds when measuring multiple qubits, where we can observe  $2^N$  total outcome probabilities depending on the states of the measured qubits.

The probabilistic nature of measurement is frequently addressed by executing a circuit many times (called the number of shots) and counting how frequently each possible output is observed. For some algorithms, the most frequent output is then taken as the correct output for the circuit, while for others, the entire output distribution is used for estimating results. However, this becomes much more difficult when errors are present in the execution of the circuit. Differentiating between variation in measurement and the effects of error at the point of measurement is difficult if not impossible, and error-correcting methods are not feasible on NISQ systems due to their small size. These errors may make it more difficult to identify a clear and convincing result from the circuit output. In the worst cases, errors can be severe enough to change which outcomes have the highest counts, thus leading observers to the wrong conclusion about the correct output.

To further complicate the evaluation of quantum circuits, a circuit can have two or more correct outputs with near equal

counts without any influence from errors. The most simple example is measuring a single qubit in the  $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$  state, commonly generated by a single Hadamard operation on a single qubit. The probability of measuring 0 or 1 is exactly  $\frac{1}{2} = \alpha^2 = \beta^2$ . We would thus expect equal counts for both outcomes. It can be difficult to evaluate the correctness of circuits with these forms of outputs. We describe our approach for evaluation and comparison in Section III.

### C. NEURAL NETWORKS

Neural networks are a powerful tool for machine learning that act as universal function approximators. The fundamental component of neural networks is the *perceptron*, an artificial neuron that computes a value output by applying an *activation function* to a weighted sum of its inputs. Given a set of input and output pairs, the perceptron aims to adjust its weights to minimize a loss function. The choice of activation and loss functions are application specific, although there are some well-known and commonly used functions for most applications. Activation functions commonly act as some form of threshold function or a transformation of the weighted sum. Examples include a simple linear function, a binary threshold function, the sigmoid function (important due to its differentiable nature compared to the standard binary threshold), or the rectified linear unit (ReLU) function, which outputs zero if the input is negative and acts as a linear function if input is positive.

Neural networks are created by combining multiple perceptrons together. While single-layer neural networks have a fair amount of discriminatory power, it is more common to stack multiple layers of perceptrons, where deeper layers receive the outputs of previous layers as inputs. The addition of multiple layers can greatly improve the performance of networks at the cost of increased training time and larger training data requirements. For example, the well-known AlexNet contains eleven total layers (including three pooling layers) [26]. Our architecture, as shown in Fig. 7, is similarly built using a number of stacked layers. The GNN layers will be explained in detail in the following section. *Flatten* layers simply compress a multidimensional array of size  $N \times N$  into an  $N^2 \times 1$  vector and are largely negligible when discussing time complexity. *Concat* layers concatenate multiple vectors into one large vector and are similarly negligible. *Dense* layers represent fully-connected layers, where every input is fed into each neuron in the layer. It follows that dense layers have a time complexity of  $\mathcal{O}(NN_L)$ , where  $N$  is the number of inputs to the layer, and  $N_L$  is the number of nodes within the layer. Finally, *argmax* is an operation that outputs the index associated with the maximum value in the input vector. In our case, *argmax* is used to identify which class action results in a maximum reward.

### D. GRAPH NEURAL NETWORKS

GNNs are a relatively new network architecture in the neural network toolkit [39]. They are specialized in handling and interpreting graph-based data that may normally be difficult for standard feedforward networks or convolutional

networks. GNNs are useful for the selection and prediction of edges and nodes, learning condensed representations of a graph as a whole, locating particular subgraphs, specialized graph traversals, and other applications. They are particularly powerful when the data naturally has a graph representation where features of both the nodes and edges are important for making decisions.

GNNs operate by sharing and diffusing information from node to node across the edges. Given an input graph, a GNN layer will compute a new representation for each node (and possibly edge) based on the values of nodes and edges within the immediate neighborhood of the node, as shown in Fig. 3. The function is typically a weighted linear combination of the node and edge features, where the weights are learned throughout the training process. This can be followed by an activation function similar to standard dense layers. Normally the neighborhood is the set of all nodes within one distance of the node in question, although this can be defined and restricted as necessary for a given problem. Multiple stacked GNN layers thus expand the neighborhood of a node, where the maximum distance of the neighborhood is equal to the number of stacked GNN layers. More layers effectively create a stronger diffusion of information across the graph. While this can be beneficial to share information, it has been observed that too many layers can decrease the performance of models containing GNNs as every node tends to approach the same representation, an average of the graph as a whole. This state destroys the individual identity of each node and negatively impacts the performance of further processing. The number of recommended GNN layers varies on the problem but is generally from one to three layers depending on the size of the graph.

GNNs can be simplified by representing the layer operations as a series of matrix multiplications. One can make assumptions on the input graph to loosen the restrictions for convergence present in the original design of GNNs. The forward diffusion operation simply becomes a multiplication of the graph's normalized adjacency matrix, the node feature matrix, and the learned weight matrix, as shown in [25]

$$X^{(k)} = \sigma(\tilde{A}X^{(k-1)}W). \quad (1)$$

Here,  $X^{(k)}$  is the node representation matrix,  $\tilde{A}$  is the renormalized adjacency matrix,  $W$  is the learned weight matrix, and  $\sigma$  is an activation function, commonly ReLU. These matrix operations demonstrate that the computational complexity of the GNN layers is  $\mathcal{O}(N^2)$ , where  $N$  is the number of nodes in the graph. This assumes that the number of nodes is greater than the number of features per node, which is a reasonable assumption at larger graph sizes.

From here, the process is further simplified by the observation that multiple layers are simply a repeated multiplication of the node matrix with an adjacency matrix, as the multiple weight matrices can be consolidated into one during the learning process. Ultimately, one can perform the work of multiple GNN layers by simply applying an activation function to the product of a power of the adjacency matrix,

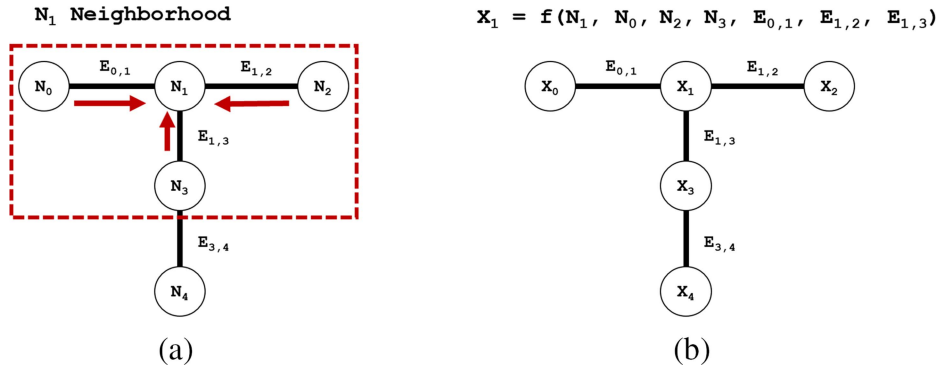


FIGURE 3. GNN update of node  $N_1$  as a function of the node and edge values within its neighborhood. (a) Initial Graph. (b) After 1 GNN Layer.

the node matrix, and the learned weight matrix, as shown in [50]

$$X^{(k)} = \sigma(\tilde{A}^k X^{(0)} W). \quad (2)$$

Here,  $X^{(0)}$  denotes the original node matrix. All other variables are the same as in (1).

Other work has been done to further enhance the applicability of GNNs to more complex types of graphs. The most relevant for our work is the addition of edge features [17]. By replacing the adjacency matrix with an edge matrix  $E$  where  $E_{i,j}$  equals the weight from node  $i$  to node  $j$ , the GNN can incorporate edge features while maintaining a simple design. This can be extended if the edge has multidimensional features by extending the dimensions of  $E$ . It is recommended to normalize the matrix using double-stochastic normalization to accelerate training. There are additional modifications that can be made to account for directed graphs by utilizing two concatenated edge matrices.

GNNs can also be combined with other neural network architectures depending on the problem at hand. We can think of the GNN operation shown in (1) as being in two steps: an initial propagation stage  $S = \tilde{A}X^{(k-1)}$ , followed by a linear inference stage  $X^k = \sigma(SW)$ . This linear stage can be replaced with other structures to solve a greater range of problems. Work has been done to demonstrate recurrent GNNs with gated units (GGCNs) [10], [30] and even attention-based GNNs [46], [47]. While these architectures are not used in this work, the recurrent nature of these structures may be beneficial for future works solving the qubit routing problem.

### E. REINFORCEMENT LEARNING

Reinforcement learning is a form of unsupervised learning that solves problems by exploring and receiving feedback from the problem environment. An agent is allowed to observe the current state of the environment and choose an action to take, changing the state of the environment and receiving some reward. Through the learning process, the agent aims to maximize the total reward earned before reaching

some terminal state of the environment. Defining a reinforcement learning problem involves describing a set of components: the set of actions an agent can take, the description of the environment (including its state and state transitions), the reward function for taking actions, the method for choosing actions, and the method for learning to maximize rewards.

The actions, environment, and rewards are directly dependent on the problem at hand. For example, if the goal is to find the shortest path in a grid-tiled environment, the actions would be the set of movements the agent can take (moving up, down, left, right) while the state of the environment would be the current position on the grid represented in  $(x, y)$  coordinates. The rewards may vary based on whether the state is a terminal or nonterminal state. Following the same example, for nonterminal states, the reward could be dependent on the distance from the current position to the end tile while the terminal state could provide a large flat reward.

By comparison, the decision and learning methods are more general. The most commonly used decision and learning method is based on Q-learning [48] and the Bellman optimality equation [7]. The goal is to provide an estimation of the reward for each potential action given the current state. From these estimations, it is common to simply select the action with the greatest estimated reward, observe the actual reward, and adjust the estimation for the (state, action) pair. These values are commonly tracked using a Q-table, containing values for every possible (state, action) pair. The simulation of the problem should then be run many times to converge to accurate reward estimations and thus accurate solutions to the problem.

As the number of states and actions grows larger, the Q-table becomes too large and unreasonable. Modern approaches instead use a neural network to learn the  $(state, action) \mapsto reward$  function, known as a Q-network. These Q-networks can be designed depending on the environment and the problem at hand, although they generally follow a certain structure—receiving the current state of the environment as input and providing a score for each possible action as output. These networks are then trained via standard

back propagation using the error between the estimated and observed rewards.

### III. MOTIVATION

We utilize IBM’s Qiskit API [38] to investigate the current performance of qubit allocation methods. Qiskit natively contains four different allocation methods: 1) trivial, 2) dense, 3) noise-adaptive [34], and 4) sabre [29]. The four methods address the mapping problem using very different approaches. The trivial layout simply maps the virtual qubits ( $q_1, q_2, \dots, q_n$ ), in order, to the physical qubits ( $0, 1, \dots, N$ ). The dense layout identifies highly connected subgraphs of the mesh and places qubits in these areas. The noise-adaptive layout is the first to rely on the most recent backend configuration data, aiming to utilize the most reliable two-qubit connections available. The sabre method utilizes an iterative process to fully route the circuit to find the final layout, then reversing the circuit using the previous final layout as a proposed initial layout. This process is repeated several times to minimize the number of necessary SWAP operations.

We tested these four layout methods on IBM’s 7-qubit *ibm\_nairobi* backend using 3-qubit to 7-qubit quantum phase estimation (QPE) circuits. We limit to a maximum of seven qubits as access to larger machines is limited. To evaluate their effects, we first run a trial of each circuit using Qiskit’s simulator with no error involved to attain a theoretical flawless outcome that we use as the ground truth for every circuit. While the measurements of the qubits are probabilistic in nature, we execute all trials with 10 000 shots to minimize the random influence. We then execute the six test circuits on the *ibm\_nairobi* backend using each layout method during compilation, again using 10 000 shots. All other compilation settings were kept default, including the routing methods. We then compared the resulting output distribution with the ground truth distribution by computing the fidelity between them. The fidelity acts as a similarity metric between the perfect ground-truth state and the real output state provided by the physical backend. A higher fidelity (bound [0, 1]) indicates a higher similarity between states.

For ease of computing fidelity  $F$ , we rely on the Hellinger distance formula described as follows:

$$F = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^N \left( \sqrt{p_i^{GT}} - \sqrt{p_i^T} \right)^2}. \quad (3)$$

Here,  $N$  is the total number of observed outputs,  $p_i^{GT}$  is the probability of output  $i$  for the ground truth distribution, and  $p_i^T$  is the probability of output  $i$  for the test distribution. The results of these trials are shown in Fig. 4. In addition to the previous four allocation methods, we also exhaustively search for the layout that achieves maximum fidelity, labeled *best* in Fig. 4. We provide the *best* outcome to compare the performance of each allocation method to the best possible outcome. Note that this exhaustive search is only feasible with a small number of qubits, as the number of

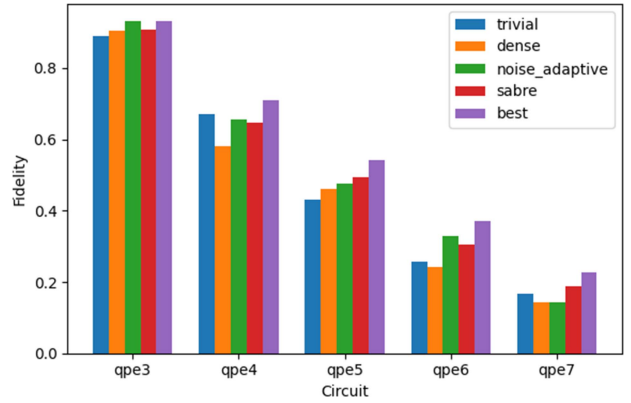


FIGURE 4. Fidelity of Qiskit’s four qubit allocation methods on the 3–7-qubit QPE algorithm after execution on *ibm\_nairobi*.

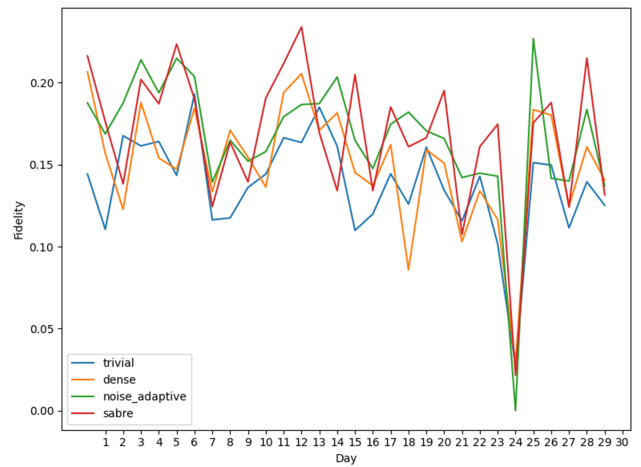
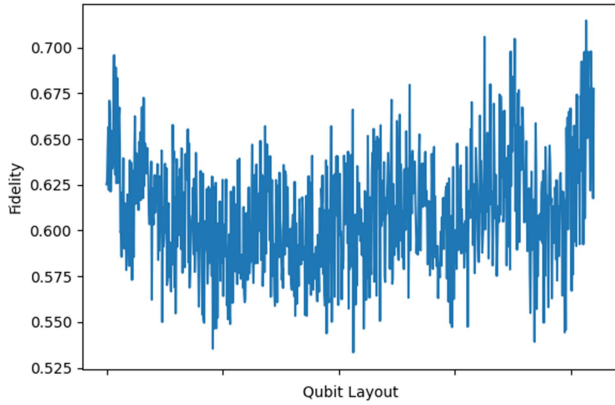


FIGURE 5. Fidelity of 7-qubit QPE when compiling with Qiskit’s four allocation methods across one month of backend configurations for *ibm\_nairobi*.

possible layouts grows extremely quickly as the number of qubits increases. As shown, we find that the layout method closest to the *best* is inconsistent. Across all of the tested circuit sizes, we see situations where each of the trivial, noise-adaptive, and sabre methods are the closest to the experimentally identified *best* fidelity.

To provide more insight into the differences between layout methods, we evaluated every layout on one month of daily calibration data for *ibm\_nairobi*, as shown in Fig. 5. This allows us to see how each allocation method performs relative to one another over time. As expected, the best allocation method is frequently either the noise-adaptive or sabre layout methods. However, the accuracy improvements are inconsistent, and we frequently see changes between which is best over time. Occasionally, they are even outperformed by the dense or trivial layouts.

It is not expected for the choice of layout to completely remove all errors and achieve a fidelity of nearly 1. However, we did expect more improvements in their behavior. To investigate the full impact of the initial layout on the outcome error and provide a metric for comparison, we provide the full



**FIGURE 6.** Fidelity of all possible 4-qubit layouts on *ibm\_nairobi's* calibration from 01-07-2022. Results are for 4-qubit QPE.

distribution of different layouts in Fig. 6 for 4-qubit QPE. It is clear that no layout is perfect, although there is a large difference between the best (above 0.7 fidelity) and worst (below 0.55 fidelity) layouts, demonstrating the importance of choosing an initial layout. Additionally, the four allocation algorithms commonly fail to identify the best layout and frequently do not even choose one of the better-than-average layouts. In total, these experiments demonstrate two main points: 1) the choice of initial layout can have a considerable impact on circuit fidelity and 2) preexisting methods are inconsistent in choosing effective layouts. There is room for improvement when selecting layouts to reduce vulnerability to error.

#### IV. ARCHITECTURE AND DATA REPRESENTATION

To improve the performance of current layout methods, we look to use GNNs as the quantum backends are naturally represented in a graph form. We combine GNNs with additional feedforward layers to predict optimal layouts given the backend error properties and an input circuit. The following sections discuss our network architecture in detail, including two main areas: 1) the backend graph input representation and processing and 2) the circuit input representation and processing. Details of the state vector and output actions are found in Section V. The overall architecture is shown in Fig. 7. Although training time is dominated by the time spent executing simulations of the quantum hardware, the overall complexity of the network layers during feedforward operations is dominated by the GNN layers with a complexity of  $\mathcal{O}(N^2)$ .

##### A. BACKEND REPRESENTATION AND PROCESSING

The superconducting backends are commonly represented as a graph, as shown in Fig. 1(b), where each node is a physical qubit with a set of properties, such as the single-qubit gate error rates, frequencies, and measurement errors. The edges, representing available CNOT connections, are weighted by the associated CNOT error rates. This configuration naturally

**TABLE 1.** List of All Node Features Collected From Physical Backends

Feature	Description
$E_{ID}$	Identity gate error
$L_{ID}$	Identity gate length
$E_{RZ}$	RZ gate error
$L_{RZ}$	RZ gate length
$E_{SX}$	SX gate error
$L_{SX}$	SX gate length
$E_X$	X gate error
$L_X$	X gate length
$T_1$	Relaxation time
$T_2$	Dephasing time
$F$	Qubit Frequency
$E_M$	Measurement error
$P_{01}$	Probability measure 0 as 1
$P_{10}$	Probability measure 1 as 0

lends itself to the edge-aware GNN variants. A sample backend with example properties is shown in Fig. 8(a).

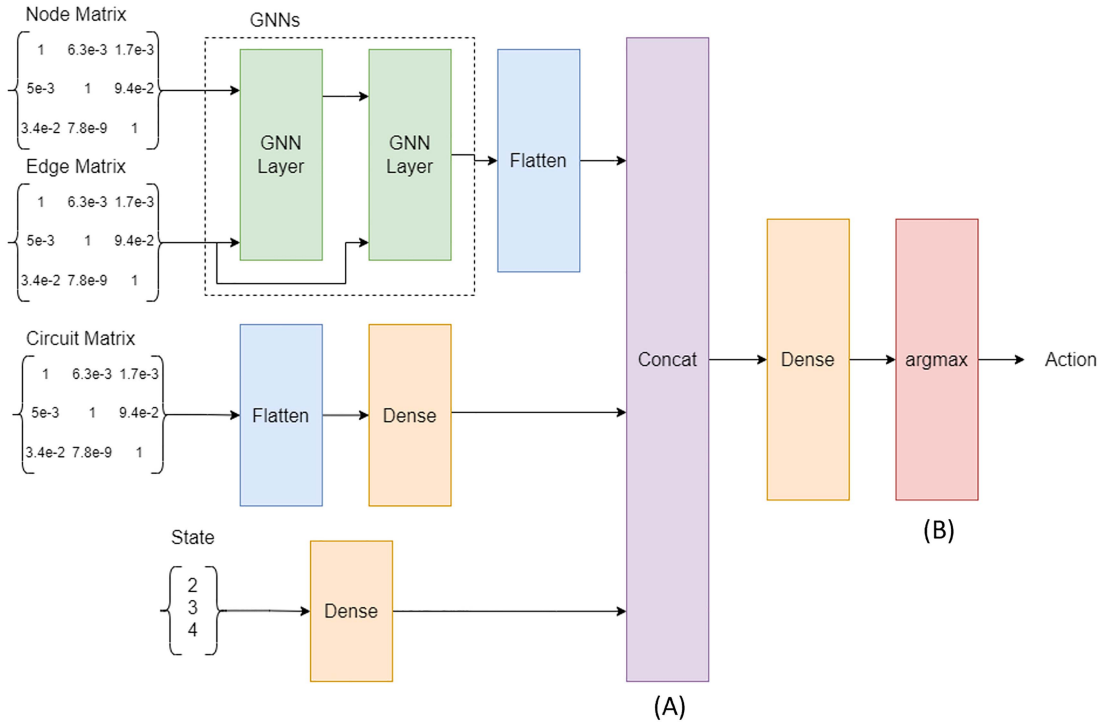
To prepare the backend for the GNN layers, we must construct both a node and edge matrix (which replaces the adjacency matrix in standard GNN). For the node matrix  $X$ , we collect several properties from each node and arrange the matrix where each row holds the properties of an individual node, as shown in Fig. 8(b). The total set of properties that we collect is found in Table 1, totaling 14 different error rates and gate lengths. The final size of the node matrix is thus  $N \times 14$ , where  $N$  is the total number of physical qubits in the backend. We access these properties using Qiskit's IBMQ provider API. The set of single-qubit gate data we collect varies depending on the basis set of gates, although all of the backends we test contain the same basis set. We choose to scale the qubit frequency as it is many orders of magnitude larger than the other values. We then normalize the matrix by row to accelerate convergence.

The edge matrix  $E$  takes the same form as a weighted adjacency matrix, where  $E_{i,j}$  equals the CNOT error between qubit  $i$  and qubit  $j$ , as shown in Fig. 8(c). Although it is not required that the CNOT error be symmetrical on all hardware implementations, we found that, for the backends we tested, the error rates were always symmetrical. We then normalize the edge matrix in a doubly-stochastic manner, following the design in [17] to ensure that both the rows and columns of  $E$  sum to 1 again aid in convergence. Given that the edge matrix is a variation of the adjacency matrix, its final dimensions are  $N \times N$ .

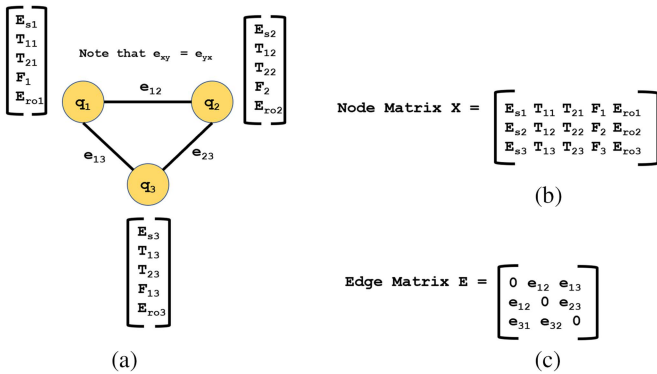
These two matrices are then fed into the network, specifically into two stacked GNN layers. Together these layers generate a new representation of the graph, which is then passed through a flattening layer to reshape the representation in preparation for concatenation with the processed circuit matrix. The GNN layers perform an edge-aware version of the forward computation described in

$$X^{(k)} = \sigma \left( \tilde{E} X^{(k-1)} W \right). \quad (4)$$

Here,  $E$  is our previously described edge matrix while  $\sigma$  is the ReLU activation function.



**FIGURE 7.** Overall architecture of the layout prediction system. (A) marks the point of concatenation between both the circuit and graph subnetworks, while (B) marks the end of the prediction network and the transition to the decoder.



**FIGURE 8.** (a) Example of 3-qubit backend with five sample node features. (b) The resulting node matrix from the backend. (c) The resulting edge matrix from the backend.

### B. CIRCUIT REPRESENTATION AND PROCESSING

To provide the circuit information to the prediction network, we first prepare a matrix containing hand-picked features to capture the behavior of the circuit. After testing a variety of different combinations, our final decision on circuit features is shown in Table 2. We believe that capturing the single-qubit operations each qubit, the measurement status of each qubit, the count of CNOT operations, and a set of CNOT partners for each qubit is sufficient for most basic circuits. We provide results in Section VII that show the influence of different look-ahead values, although by default we only use the first CNOT partner. Currently, this representation would

**TABLE 2.** List of All Circuit Features Collected From Test Circuits

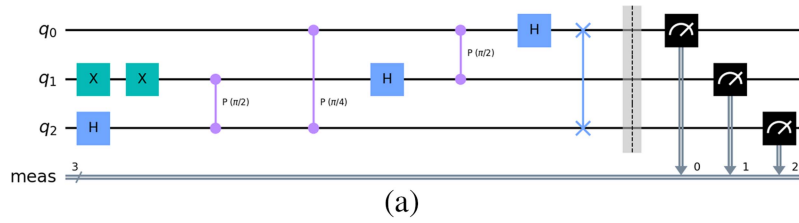
Feature	Description
$N_{ID}$	Number of involved identity operations
$N_{RZ}$	Number of involved RZ operations
$N_{SX}$	Number of involved SX operations
$N_X$	Number of involved X operations
$N_{CNOT}$	Number of involved CNOT operations
$M$	Measurement status
$T_i$	$i$ th partner qubit for CNOT operations*

likely fail to represent more complex circuits involving mid-execution measurement and qubit reset, although these operations do not occur in any of our test circuits and are not commonly found. An example circuit and associated circuit matrix are shown in Fig. 9. We simplify the example by counting all single-qubit operations as one feature rather than individual single-qubit operation counts.

It is important to note that we do not use the original logical circuit to prepare these representations, as they may change through the steps of the compilation process before preparing a layout. The most important changes that can occur are decomposing multiqubit operations and subcircuits and mapping to basis gates, as these can greatly change the view of which operations the circuit performs. Instead, we acquire the intermediate circuit during the compilation process at the point where qubit mapping normally occurs, after these other operations. This allows us to represent the circuit as accurately as possible for choosing a layout.

Once this feature matrix is created for the circuits, we can feed them to a flattening layer dense layer that reshapes the





Circuit Matrix C =

# Gates	Measured	1 <sup>st</sup> CNOT Partner
5	1	2
6	1	2
5	1	1

(b)

FIGURE 9. (a) Example of 3-qubit QFT circuit. (b) Constructed circuit matrix for the example circuit.

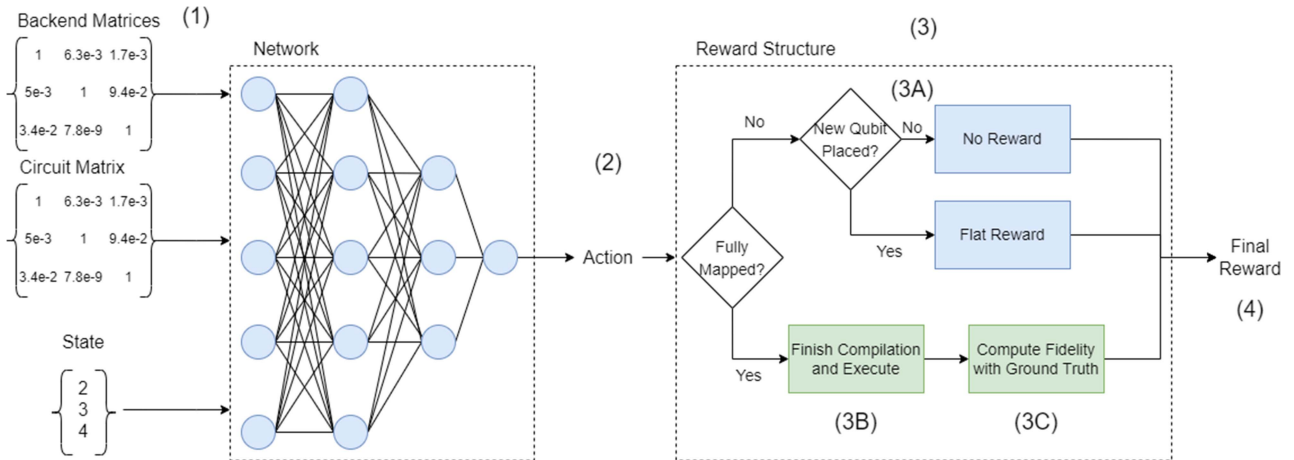


FIGURE 10. Training process data flow of the prediction network.

representation similar to that of the graph matrix after the GNN layers, as shown in Fig. 7. This representation passes through a dense layer to adjust to circuit features. The two representation vectors are then concatenated along with the current state vector and transposed before being passed to another dense layer that now operates on the complete data of both learned representations of the backend and the circuit. This layer provides a score (the proposed  $Q$ -value) for each possible action that can be taken during allocation. We then identify the maximum score associated with the best action to take at the current compilation step and execute that action during the given iteration.

### V. REINFORCEMENT LEARNING SETUP

In this section, we describe the components of the reinforcement learning process, namely the actions, environment, rewards, and training process. The overall training process can be found in Fig. 10.

### A. ACTIONS

When mapping the qubits, the available actions are simply one placement action for each (*logical, physical*) qubit pair. This placement action represents assigning the logical qubit to the associated physical qubit for the initial layout. To account for circuits with fewer logical qubits than the available physical qubits, we extend the logical qubits with ancilla qubits to equal the number of physical qubits. The full mapping is done to provide a fixed-length input vector to the neural networks. This allows the network to learn mappings for any number of logical qubits up to the number of physical qubits in the machine. In total, this results in  $N_{phys}^2$  actions. This also characterizes the total number of outcomes resulting from the final dense layer in Fig. 7. Following an Epsilon-Greedy policy, with  $\epsilon = 0.05$ , we select the action associated with the maximum predicted value with probability  $1 - \epsilon$  and a random action with probability  $\epsilon$  to drive our training decisions.

## B. ENVIRONMENT

To define the environment, we first represent the state of the physical hardware and the circuit as described in Section IV. This requires both an edge and node matrix from the physical backend that describes the error characteristics from the latest calibration and a circuit matrix that represents the operations that must take place for a given circuit.

These inputs are then complemented with a vector containing the current mapping of qubits, specifically mapping from *physical*  $\rightarrow$  *logical* qubits. This captures the current state of the layout, specifically a snapshot of the current layout at a given time during compilation. The vector is initialized to all zero values, indicating no qubits have been placed, and gradually fills with nonzero values as placement actions are taken each iteration. Together, the matrices and state vector capture the problem itself as well as the current intermediate solution.

## C. REWARDS

When providing rewards, we first consider the placement of ancilla qubits. As these qubits are not important to the execution of the circuit, placing the qubits provides no reward. Similarly, when attempting to place a qubit that has already been assigned to a physical qubit, no reward is given. In contrast, placing a previously unplaced logical qubit provides a flat reward to encourage prioritization.

The most interesting case is the reward given when completing the mapping of all logical qubits. In this case, we first execute the circuit on the simulator using the error profile of the backend. We choose to use the simulator as we do not have dedicated access to a physical backend for training. We then compare the output distribution to an error-free output distribution that acts as our ground truth. This error-free distribution is obtained by executing the circuit on a simulator with no error simulation. This is effectively a theoretically perfect outcome for the circuit, providing a target for comparison.

To provide a tangible value, we compute the Hellinger fidelity between the two distributions, as shown in (3). The more similar the output distributions are, the closer this value approaches 1. This is then scaled by 100 and provided as the final reward. This guides GNAQC target configurations that are most similar to the error-free distribution.

## D. TRAINING

To train the network, we rely on the Qiskit Aer simulator to simulate the execution of the circuit using the proposed mapping. We then compute the fidelity between the results of the simulation and the ground truth output of the circuit as previously discussed. Once again, we rely on the Hellinger fidelity, as described in (3), as our reward metric as opposed to success rate as we do not necessarily know the correct output of the given circuit with which to compute a success rate. To make the approach more general, we instead target the entire ground truth distribution using the fidelity. This error

is then used for back-propagation for training the network as a whole.

The full training process is shown in Fig. 10. First, the processed edge, node, and circuit matrices are fed to the prediction network in step (1). The network outputs a suggested action to take, namely a qubit placement, in step (2). The reward for this action is calculated in step (3), where the value of the reward depends on the result of the action. If the action results in a fully-mapped circuit, we finish compilation (routing and final optimization) and simulate the final circuit in step (3B) using Qiskit's Aer simulator. The simulator is prepared with a noise model built on the error properties of the collected backend under test. In step (3B), we collect the output counts from the simulator and compute the fidelity with the ground truth distribution. If the action did not result in a fully-mapped circuit, we instead give either a reward of 0 if the qubit was already placed or a flat 10 if the qubit is newly placed. Finally, we use this reward for the update process following the typical Q-learning update rule in step (4).

It is worth noting here that we do not need to rely on the simulator, which will not be feasible for increasingly large circuits and backends, during this training process. We could rely on other success metrics, like estimated success probability [8] or other future methods, which may be more scalable. However, we chose to use the simulator to be more accurate to the hardware. Ideally, one would have dedicated access to a physical machine for the training process, which would address both the accuracy and scalability concerns. A ground truth distribution would still need to be computed either analytically, through simulation (which may not be feasible), or an averaging of many runs on the hardware to approach a stable distribution. New methods, such as splitting the circuit to improve fidelity, may aid in this process [44]. If a ground truth distribution cannot be identified, the reinforcement learning process would need to be modified to rely on other metrics that are correlated with fidelity, such as gate count.

## VI. DATA COLLECTION AND EXPERIMENTATION

Throughout our experimentation, we rely on a set of various test circuits at different sizes executed on several different physical backends. We focus on a set of six different circuits as mentioned previously in Section III: the Deutsch–Jozsa (DJ) algorithm, the Bernstein–Vazirani (BV) algorithm, Simon's algorithm, the quantum Fourier transform (QFT), the QPE algorithm, and Grover's search algorithm. We prepare these circuits using 3, 4, 5, 6, 7, 15, and 27 qubits. We believe that two qubits are simply too trivial, and we are limited to evaluating on backends with 7 or 27 maximum qubits. The characteristics of these circuits, specifically the count of the final gates used for each algorithm, at each circuit size are detailed in Table 3.

For the backends, we collected calibrations for *ibm\_nairobi*, a 7-qubit backend, and *ibm\_algiers*, a 27-qubit backend. We selected these two as a sample of the available 7-qubit and 27-qubit machines we can access.

**TABLE 3. Details of the Six Test Circuits at All Sizes Used for All Experiments**

Name	Description	#Qubits	#1Q	#2Q
<i>DJ</i>	Deutsch-Jozsa	3	30	12
		5	128	60
		7	524	248
		15	6584	3152
		27	21880	10398
<i>BV</i>	Bernstein-Vazirani	3	30	12
		5	128	60
		7	524	248
		15	6584	3152
		27	21880	10398
<i>Simon</i>	Simon's Algorithm	3	26	12
		5	128	60
		7	484	248
		15	6328	3152
		27	18468	10398
<i>QFT</i>	Quantum Fourier Transform	3	17	9
		4	28	18
		5	43	26
		6	224	78
		7	457	208
		15	2076	1226
		27	16937	3688
<i>QPE</i>	Quantum Phase Estimation	3	21	8
		4	43	20
		5	80	42
		6	240	98
		7	483	244
		15	2206	1486
<i>Grover</i>	Grover's Search Algorithm	3	98	30
		4	202	102
		5	459	256
		6	905	428
		7	1820	768
		15	10238	5820
		27	89374	20838

**TABLE 4. Details of the Two Backends Used for Collecting Data for All Experiments**

Name	# Qubits	Basis Gates	Topology
<i>ibm_nairobi</i>	7	CX, ID, RZ, SX, X	I
<i>ibm_algiers</i>	27	CX, ID, RZ, SX, X	Square

We specifically collected the archived daily calibrations from January 1st, 2022, through the end of May 2022. The backends vary in topology, with *ibm\_nairobi* having an I shape and *ibm\_algiers* having an adjusted square shape. Both backends share the same set of basis gates. These details are summarized in Table 4.

## VII. RESULTS

To evaluate the general performance of GNAQC, we predict layouts for the circuits using the most recent calibrations at their time of execution. The historical backend calibrations are used for training. We compare these results to the previously measured errors for the four layout methods contained within Qiskit. These results are shown in Fig. 11.

Here, the “learned” layout is the behavior of the layout output by our method. It can be observed that the learned layout generally outperforms the preexisting layouts for each

benchmark at different algorithm sizes. The learned layouts consistently perform better on simpler algorithms, such as DJ, BV, and Simon. There is less, though fairly consistent, improvement on the larger algorithms. On average, however, we see a relative improvement in fidelity of approximately 12.7%.

We group the data by each circuit regardless of backend or qubit size to inspect the mean performance on the individual algorithms. These results are shown in Fig. 12. Here, the learned layouts show improvement or consistent behavior on most circuits. In the worse cases, GNAQC performs comparably to the best alternative layout method. The magnitude of the improvement varies from circuit to circuit and is also dependent on the next-best choice. In general, we believe this variation is due to the effect different layouts have depending on the length of the algorithm, where shorter circuits are simply more influenced by the initial position of qubits, while longer algorithms are likely more influenced by the routing methods.

Next, we group the results by the number of qubits involved in the algorithm to observe performance based on the size of the circuits, as shown in Fig. 13. As we can see, the largest improvement is found on smaller circuit sizes. We see the most variation in behavior among the layouts at 3–5 qubits, with more consistent performance among all five methods at larger sizes. We identify two main reasons for this variation in behavior. First, as the depth of the circuit increases due to the increased number of qubits, the fidelity decreases drastically. This results in less room for the layouts to vary as the fidelity is simply so low. Second, we believe that this has to do with the percentage of qubits used on the backend and the topology of the machine itself. When using all of the qubits on the machine, more SWAPs will likely need to be added to allow the circuit to function regardless of the initial position of qubits. At smaller sizes, particularly three qubits, the number of added SWAPs may vary greatly based on the initial position of qubits. It may be possible to place them in a configuration where no SWAPs are necessary, such as a triangle section in the mesh, or to place them at opposite sides of the mesh where many SWAPs are necessary.

*Look-ahead:* In all of the previous results shown, we have utilized a CNOT look-ahead window of length 1, as described in Section IV-B. However, we also examined a variable look-ahead window from lengths 1 to 5. For a chosen look-ahead value  $LA$ , our circuit matrix will hold the first  $LA$  CNOT partners for each qubit. We evaluated our method for all  $1 \leq LA \leq 5$  for 7-qubit circuits using a simulator based on the properties of *ibm\_nairobi*. We chose to use the simulator here due to time restrictions and the number of executions necessary. The results are shown in Fig. 14. As shown, the value of  $LA$  does not have a large impact on performance. This could be due to insufficient training data when increasing the number of parameters.

To evaluate the cost of GNAQC, we measure the execution time of each layout method for QFT as shown in Fig. 15. GNAQC-onerun is the time to perform a single inference

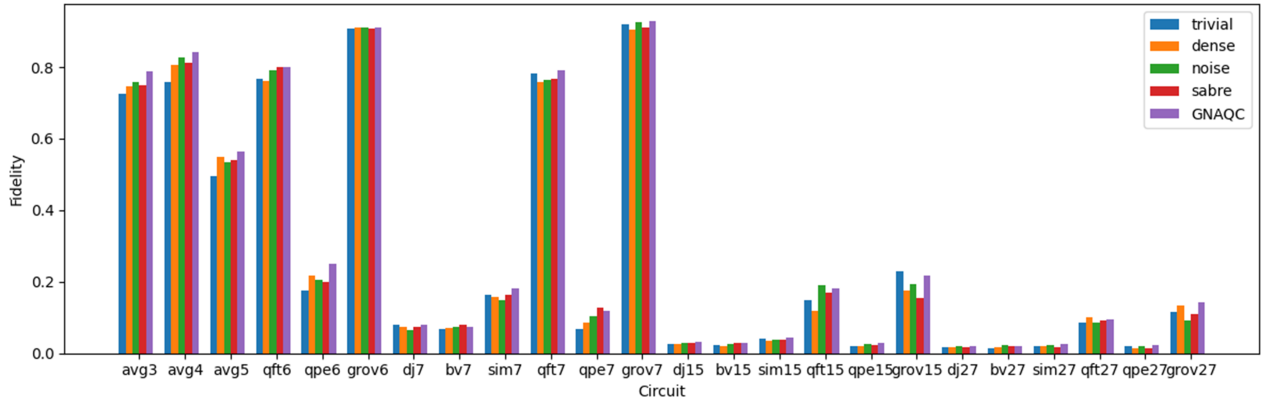


FIGURE 11. All results from testing 3–27 qubit algorithms on ibm aljers and nairobi when compiling with each of the five allocation methods.

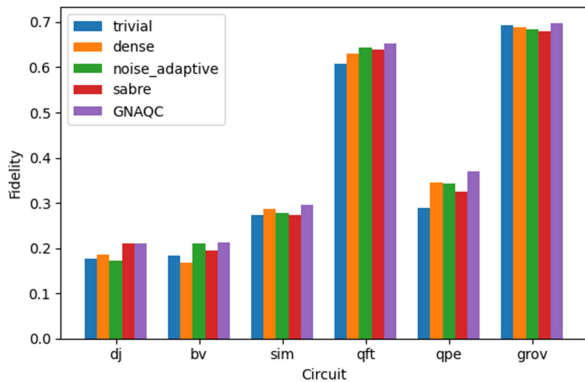


FIGURE 12. Fidelity of different layout methods grouped by each test circuit.

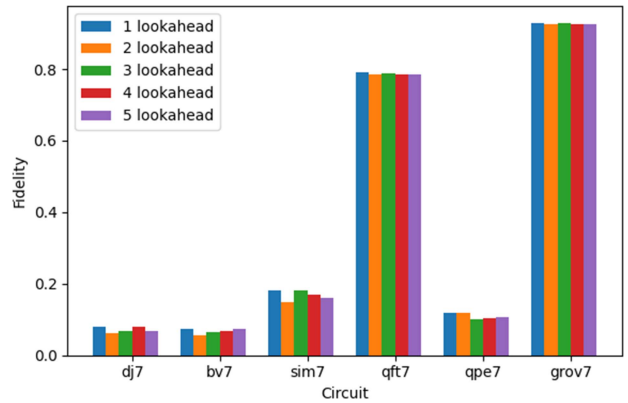


FIGURE 14. Fidelity for 1–5 look-ahead representations for the test circuits.

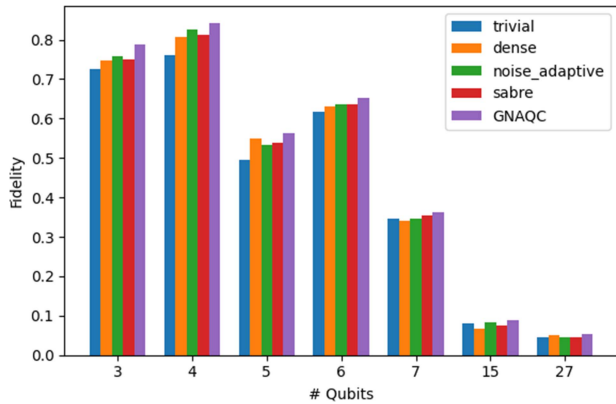


FIGURE 13. Fidelity of different layout methods grouped by number of qubits used in the circuit.

stage while GNAQC-total is the total time to place all qubits (one inference iteration per qubit). It is clear that GNAQC performs in line with the other methods and scales better to larger qubit sizes than the sabre method.

## VIII. RELATED WORKS

### A. IMPROVING ACCURACY

Increasing circuit resilience to errors is a major field in quantum computing research. A common approach involves modifying compilation, either the allocation or routing passes.

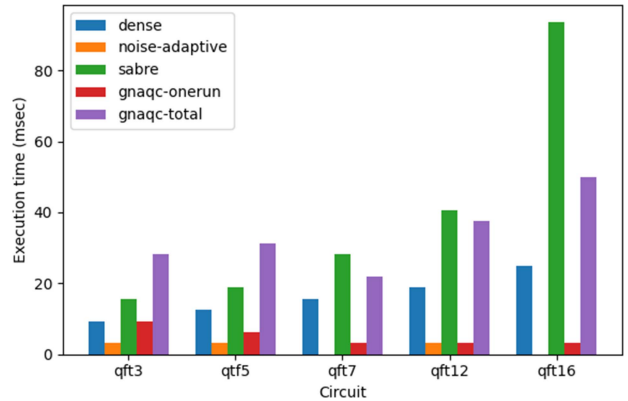


FIGURE 15. Compilation time for each of the layout methods. The trivial method is not included due to negligible execution time.

The first major works identified that routing should focus not only on the number of inserted SWAP operations but also the reliability of the qubit connections [34], [45]. This included a modified routing method to be aware of the CNOT error rates. By focusing on using the most reliable qubits and connections, the final accuracy of the circuit could be improved substantially. Later observations showed that performing CNOT operations in parallel with other nearby operations would increase error rates, suggesting that routing

methods should plan for this cross-talk error and attempt to avoid parallel operations on adjacent qubits where possible [13], [35]. While this creates a more complex routing problem, isolating CNOT operations where possible also considerably improves circuit performance. Others have demonstrated that one can improve reliability by executing a circuit in multiple parts and then reconstructing the overall distribution [44]. This allows for focusing on only a few qubits at a time and using only the most reliable physical qubits while avoiding unnecessary cross-talk then combining the individual trial outputs into a final, more accurate distribution. Meanwhile, the research on debugging, protecting, and reusing resources is also trying to find solutions to mitigate the constraints of error in the field. The quantum assertion technique was proposed and evolved in [22], [31], and [32] to locate the errors and bugs while running quantum algorithms. Applying QEC to superconducting quantum chips is also being actively studied in [6], [23], [19], and [20]. While hardware implementations of error correction will likely be valuable in the future, current NISQ sizes struggle to implement them while maintaining enough usable qubits for practical algorithms. Reusing the valuable quantum resources, the physical qubits, is being studied in [14], including resetting qubits to basis states or reversing operations to reduce the total number of qubits or SWAPs necessary during execution.

## B. QUBIT ALLOCATION

The two most relevant allocation methods are the two already contained within Qiskit, the noise-adaptive [34], and sabre [29] methods. These methods are used as two comparisons to GNAQC throughout our experiments. Some methods utilize locating the optimal layout at smaller sizes to produce heuristics that are then tested on larger-scale systems [4], [52], [42]. Our demonstration in Section III follows a similar approach, where we compare the performance of each method against the best possible layouts on smaller sizes. While this is helpful, it is difficult to extrapolate from smaller to larger sizes due to growing complexity. Other works similarly search the set of possible layouts while guided by fidelity or other success methods [15], [27], [28]. All of these approaches aim to minimize the vulnerability of the circuit through the choice of an initial layout through a variety of different methods.

## C. GNN APPLICATIONS

The base GNN [39] design has many modifications for a variety of different applications. Extensions have been added to the GNN for recurrent units similar to gated recurrent networks [10], [30]. The addition of recurrent units improves performance on deep GNNs, namely when applied to a graph changing over time, similar to recurrent neural networks on traditional time series data. Graph convolutional networks (GCNs) enhance the process and improve upon the original design for node classification tasks [25], [50]. Again, GCNs borrow from traditional convolutional neural

networks, commonly used for multidimensional tasks, such as image processing and apply the approach to graphs. Several tools have been built using GCNs for learning graph representations [18], [37] while others leverage adversarial approaches for learning graph embeddings [36]. The idea of using GNNs to improve compilation has similarly been used in hardware placement for classical circuits [3], [33], [51], but not for quantum circuits.

## IX. CONCLUSION

We have proposed GNAQC, a new GNN-based neural network architecture for improving the reliability of superconducting quantum circuits by identifying more resilient layouts. We compare the proposed layouts with the preexisting layout methods contained within Qiskit and find a mean 12.7% relative increase in fidelity across both backend configurations with six different circuits. In the future, we believe we could achieve even greater results by expanding the work to include a routing method using recurrent GNNs or experimenting with different feature representations.

## REFERENCES

- [1] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018, doi: [10.22331/q-2018-08-06-79](https://doi.org/10.22331/q-2018-08-06-79).
- [2] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [3] A. Agnesina, K. Chang, and S. K. Lim, "VISI placement parameter optimization using deep reinforcement learning," in *Proc. 39th Int. Conf. Comput.-Aided Des.*, 2020, pp. 1–9, doi: [10.1145/3400302.3415690](https://doi.org/10.1145/3400302.3415690).
- [4] A. Ash-Saki, M. Alam, and S. Ghosh, "QURE: Qubit re-allocation in noisy intermediate-scale quantum computers," in *Proc. 56th Annu. Des. Automat. Conf.*, 2019, pp. 1–6, doi: [10.1145/3316781.3317888](https://doi.org/10.1145/3316781.3317888).
- [5] A. Ashikhmin and E. Knill, "Nonbinary quantum stabilizer codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 3065–3072, Nov. 2001, doi: [10.1109/18.959288](https://doi.org/10.1109/18.959288).
- [6] R. Barends et al., "Superconducting quantum circuits at the surface code threshold for fault tolerance," *Nature*, vol. 508, no. 7497, pp. 500–503, 2014, doi: [10.1038/nature13171](https://doi.org/10.1038/nature13171).
- [7] R. Bellman, "The theory of dynamic programming," *Bull. Amer. Math. Soc.*, vol. 60, no. 6, pp. 503–515, 1954, doi: [10.1090/S0002-9904-1954-09848-8](https://doi.org/10.1090/S0002-9904-1954-09848-8).
- [8] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," in *Quantum Computation and Information. ser. Contemporary Mathematics*, vol. 305. Providence, RI, USA: Amer. Math. Soc, 2002, doi: [10.1090/conm/305](https://doi.org/10.1090/conm/305).
- [9] W. Cai, Y. Ma, W. Wang, C.-L. Zou, and L. Sun, "Bosonic quantum error correction codes in superconducting quantum circuits," *Fundam. Res.*, vol. 1, no. 1, pp. 50–67, 2021, doi: [10.1016/j.fmre.2020.12.006](https://doi.org/10.1016/j.fmre.2020.12.006).
- [10] C. Chen et al., "Gated residual recurrent graph neural networks for traffic prediction," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 485–492, doi: [10.1609/aaai.v33i01.3301485](https://doi.org/10.1609/aaai.v33i01.3301485).
- [11] J. Clarke and F. K. Wilhelm, "Superconducting quantum bits," *Nature*, vol. 453, no. 7198, pp. 1031–1042, 2008, doi: [10.1038/nature07128](https://doi.org/10.1038/nature07128).
- [12] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Phys. Rev. A*, vol. 100, no. 3, 2019, Art. no. 032328, doi: [10.1103/PhysRevA.100.032328](https://doi.org/10.1103/PhysRevA.100.032328).
- [13] Y. Ding, P. Gokhale, S. F. Lin, R. Rines, T. Propson, and F. T. Chong, "Systematic crosstalk mitigation for superconducting qubits via frequency-aware compilation," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2020, pp. 201–214, doi: [10.1109/MICRO50266.2020.00028](https://doi.org/10.1109/MICRO50266.2020.00028).
- [14] Y. Ding et al., "SQUARE: Strategic quantum ancilla reuse for modular quantum programs via cost-effective uncomputation," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Architecture*, 2020, pp. 570–583, doi: [10.1109/ISCA45697.2020.00054](https://doi.org/10.1109/ISCA45697.2020.00054).

- [15] W. Finigan, M. Cubeddu, T. Lively, J. Flick, and P. Narang, "Qubit allocation for noisy intermediate-scale quantum computers," 2018, *arXiv:1810.08291*, doi: [10.48550/arXiv.1810.08291](https://doi.org/10.48550/arXiv.1810.08291).
- [16] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Phys. Rev. A*, vol. 86, no. 3, 2012, Art. no. 032324, doi: [10.1103/PhysRevA.86.032324](https://doi.org/10.1103/PhysRevA.86.032324).
- [17] L. Gong and Q. Cheng, "Exploiting edge features for graph neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9211–9219, doi: [10.1109/CVPR.2019.00943](https://doi.org/10.1109/CVPR.2019.00943).
- [18] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035. [Online]. Available: [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7e9bea9-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7e9bea9-Paper.pdf)
- [19] A. Holmes, Y. Ding, A. Javadi-Abhari, D. Franklin, M. Martonosi, and F. T. Chong, "Resource optimized quantum architectures for surface code implementations of magic-state distillation," *Microprocessors Microsyst.*, vol. 67, pp. 56–70, 2019, doi: [10.1016/j.micpro.2019.02.007](https://doi.org/10.1016/j.micpro.2019.02.007).
- [20] A. Holmes, M. R. Jokar, G. Pasandi, Y. Ding, M. Pedram, and F. T. Chong, "NISQ: Boosting quantum computing power by approximating quantum error correction," in *Proc. IEEE/ACM 47th Annu. Int. Symp. Comput. Architecture*, 2020, pp. 556–569, doi: [10.1109/ISCA45697.2020.00053](https://doi.org/10.1109/ISCA45697.2020.00053).
- [21] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, "Surface code quantum computing by lattice surgery," *New J. Phys.*, vol. 14, no. 12, 2012, Art. no. 123011, doi: [10.1088/1367-2630/14/12/123011](https://doi.org/10.1088/1367-2630/14/12/123011).
- [22] Y. Huang and M. Martonosi, "Statistical assertions for validating patterns and finding bugs in quantum programs," in *Proc. 46th Int. Symp. Comput. Architecture*, 2019, pp. 541–553, doi: [10.1145/3307650.3322213](https://doi.org/10.1145/3307650.3322213).
- [23] A. Javadi-Abhari et al., "Optimized surface code communication in superconducting quantum computers," in *Proc. IEEE/ACM 50th Annu. Int. Symp. Microarchitecture*, 2017, pp. 692–705, doi: [10.1145/3123939.3123949](https://doi.org/10.1145/3123939.3123949).
- [24] D. Kielpinski, C. Monroe, and D. J. Wineland, "Architecture for a large-scale ion-trap quantum computer," *Nature*, vol. 417, no. 6890, pp. 709–711, 2002, doi: [10.1038/nature00784](https://doi.org/10.1038/nature00784).
- [25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*, doi: [10.48550/arXiv.1609.02907](https://doi.org/10.48550/arXiv.1609.02907).
- [26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, Jun. 2017, doi: [10.1145/3065386](https://doi.org/10.1145/3065386).
- [27] T. LeCompte, F. Qi, and L. Peng, "Robust cache-aware quantum processor layout," in *Proc. IEEE Int. Symp. Reliable Distrib. Syst.*, 2020, pp. 276–287, doi: [10.1109/SRDS51746.2020.00035](https://doi.org/10.1109/SRDS51746.2020.00035).
- [28] T. LeCompte, F. Qi, X. Yuan, N.-F. Tzeng, M. H. Najaf, and L. Peng, "Graph neural network assisted quantum compilation for qubit allocation," in *Proc. ACM Great Lakes Symp. VLSI*, 2023, pp. 415–419, doi: [10.1145/3583781.3590300](https://doi.org/10.1145/3583781.3590300).
- [29] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 1001–1014, doi: [10.1145/3297858.3304023](https://doi.org/10.1145/3297858.3304023).
- [30] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2015, *arXiv:1511.05493*, doi: [10.48550/arXiv.1511.05493](https://doi.org/10.48550/arXiv.1511.05493).
- [31] J. Liu, G. T. Byrd, and H. Zhou, "Quantum circuits for dynamic runtime assertions in quantum computation," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 1017–1030, doi: [10.1145/3373376.3378488](https://doi.org/10.1145/3373376.3378488).
- [32] J. Liu and H. Zhou, "Systematic approaches for precise and approximate quantum state runtime assertion," in *Proc. 27th IEEE Int. Symp. High- Perform. Comput. Architecture*, 2021, vol. 21, pp. 179–193, doi: [10.1109/HPCA51647.2021.00025](https://doi.org/10.1109/HPCA51647.2021.00025).
- [33] Y.-C. Lu, S. Pentapati, and S. K. Lim, "VLSI placement optimization using graph neural networks," in *Proc. 34th Conf. Neural Inf. Process. Syst. ML Syst. Workshop*, 2020, pp. 6–12. [Online]. Available: [https://mlforsystems.org/assets/papers/neurips2020/vlsi\\_placement\\_lu\\_2020.pdf](https://mlforsystems.org/assets/papers/neurips2020/vlsi_placement_lu_2020.pdf)
- [34] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 1015–1029, doi: [10.1145/3297858.3304075](https://doi.org/10.1145/3297858.3304075).
- [35] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 1001–1016, doi: [10.1145/3373376.3378477](https://doi.org/10.1145/3373376.3378477).
- [36] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," 2018, *arXiv:1802.04407*, doi: [10.48550/arXiv.1802.04407](https://doi.org/10.48550/arXiv.1802.04407).
- [37] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 701–710, doi: [10.1145/2623330.2623732](https://doi.org/10.1145/2623330.2623732).
- [38] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023. [Online]. Available: <https://qiskit.org/>
- [39] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," in *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009, doi: [10.1109/TNN.2008.2005605](https://doi.org/10.1109/TNN.2008.2005605).
- [40] P. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41, no. 2, pp. 303–332, 1999, doi: [10.1137/S0036144598347011](https://doi.org/10.1137/S0036144598347011).
- [41] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A*, vol. 52, no. 4, 1995, Art. no. R2493, doi: [10.1103/PhysRevA.52.R2493](https://doi.org/10.1103/PhysRevA.52.R2493).
- [42] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proc. Int. Symp. Code Gener. Optim.*, 2018, pp. 113–125, doi: [10.1145/3168822](https://doi.org/10.1145/3168822).
- [43] A. M. Steane, "Error correcting codes in quantum theory," *Phys. Rev. Lett.*, vol. 77, no. 5, 1996, Art. no. 793, doi: [10.1103/PhysRevLett.77.793](https://doi.org/10.1103/PhysRevLett.77.793).
- [44] S. S. Tannu and M. Qureshi, "Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2019, pp. 253–265, doi: [10.1145/3352460.3358257](https://doi.org/10.1145/3352460.3358257).
- [45] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for NISQ-era quantum computers," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 987–999, doi: [10.1145/3297858.3304007](https://doi.org/10.1145/3297858.3304007).
- [46] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li, "Attention-based graph neural network for semi-supervised learning," 2018, *arXiv:1803.03735*, doi: [10.48550/arXiv.1803.03735](https://doi.org/10.48550/arXiv.1803.03735).
- [47] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*, doi: [10.48550/arXiv.1710.10903](https://doi.org/10.48550/arXiv.1710.10903).
- [48] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, 1992, doi: [10.1007/BF00992698](https://doi.org/10.1007/BF00992698).
- [49] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, no. 5886, pp. 802–803, 1982, doi: [10.1038/299802a0](https://doi.org/10.1038/299802a0).
- [50] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6861–6871. [Online]. Available: <https://proceedings.mlr.press/v97/wu19e.html>
- [51] G. Zhang, H. He, and D. Katabi, "Circuit-GNN: Graph neural networks for distributed circuit design," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7364–7373. [Online]. Available: <http://proceedings.mlr.press/v97/zhang19e.html?ref=https://githubhelp.com>
- [52] P. Zhu, X. Cheng, and Z. Guan, "An exact qubit allocation approach for NISQ architectures," *Quantum Inf. Process.*, vol. 19, no. 11, pp. 1–21, 2020, doi: [10.1007/s11128-020-02901-4](https://doi.org/10.1007/s11128-020-02901-4).