The Imitation Game: Leveraging CopyCats for Robust Native Gate Selection in NISQ Programs

Poulami Das
Georgia Institute of Technology
Atlanta, USA
poulami@gatech.edu

Eric Kessler
Amazon Braket (AWS)
New York, USA
erikessl@amazon.com

Yunong Shi
Amazon Braket (AWS)
New York, USA
shiyunon@amazon.com

***Abstract*—Quantum programs are written in high-level languages, whereas quantum hardware can only execute low-level *native* gates. To run programs on quantum systems, each high-level instruction must be decomposed into native gates. This process is called *gate nativization* and is performed by the compiler. Recent quantum computers support a richer native gate set to reduce crosstalk by tackling frequency crowding and enable compilers to generate quantum executables with fewer native gates. On these systems, any two-qubit CNOT instruction can be decomposed using more than a single two-qubit native gate. For example, a CNOT can be decomposed using either *XY*, *CPHASE*, or *CZ* native gates on Rigetti machines. Unfortunately, two-qubit native gates have high-error rates and exhibit temporal and spatial variations, which limits the success-rate of quantum programs. Therefore, identifying the native gate that maximizes the success-rate of each CNOT operation in a program is crucial.**

Our experiments on Rigetti machines show that noise-adaptive gate nativization to select the native gate with the highest fidelity for each CNOT operation is often sub-optimal at application-level. This is because the performance of such nativization heavily depends on the correctness of the device calibration data which only provides the average gate fidelities and may not accurately capture the error trends specific to the qubit state space of a program. Moreover, the calibration data may go stale due to device drifts going undetected. To overcome these limitations, we propose *Application-specific Native Gate Selection (ANGEL)*. ANGEL designs a *CopyCat* that imitates a given program but has a known solution. Then, ANGEL employs the CopyCat to test different combinations of native gates and learn the optimal combination, which is then used to nativize the given program. To avoid an exponential search, ANGEL uses a divide-and-conquer based localized search, the complexity of which scales linear with the number of device links used by the program. Our evaluations on Rigetti Aspen-11 shows that ANGEL improves the success-rate of programs by 1.40x on average and by up-to 2x.

***Index Terms*—Quantum, Gate nativization, NISQ Compilation**

I. Introduction

The demonstration of quantum supremacy on certain artificial tasks represents a significant milestone in quantum computing [5, 64, 127]. Today, quantum computers are getting to the regime where they promise to power real-world applications soon [47]. Unfortunately, qubit devices are noisy, and their high error-rates limit the probability of successful execution (or *success-rate*) of most quantum programs. For example, the average error-rate of a two-qubit operation ranges between 1-12.5% on existing quantum systems [1, 2, 43]. Although near-term quantum computers are rapidly growing in size [44, 46], they would still lack the resources required to run applications in a fully fault-tolerant manner. Instead, these *Noisy Intermediate Scale Quantum (NISQ)* [90] computers run programs in the presence of noise. Consequently, software error-mitigation policies will play a crucial in leveraging NISQ systems to accelerate practical applications [48, 78].

Quantum programs are often written in high-level languages for the ease of programmability, whereas quantum hardware only supports a limited number of *native* or *basis* gates. To run a quantum program, the compiler translates each instruction of a program into an equivalent gate sequence comprising of only native gates supported by the target device. For example, a two-qubit CNOT operation is decomposed into a combination of R_X, R_Z, and XY native gates on Rigetti Aspen devices, as shown in Figure 1(a). This process is called *gate nativization*.

A richer native gate set offers greater flexibility to the compiler to express any operation in a quantum circuit and thus, reduces the total number of native gates in the quantum executable to be run on the quantum hardware [56]. Multiple two-qubit native gates also add an extra dimension of controllability that alleviates the problem of frequency crowding [55] and crosstalk errors in highly connected qubit architectures [12, 89, 94]. As a result, many recent systems support more than a single two-qubit native gate [14, 35, 45, 50, 71, 79, 89, 95]. For example, Rigetti devices support three different two-qubit native gates, namely *XY*, *CZ*, and *CPHASE*.

Optimal nativization is crucial for NISQ applications, but non-trivial on NISQ systems that support multiple two-qubit native gates because now a compiler can choose to nativize a CNOT in various ways. As CNOT operations enable quantum computers to leverage the property of *entanglement* and create highly correlated states, they are fundamental to quantum algorithms [9]. Unfortunately, CNOT operations cannot be implemented perfectly on real quantum hardware, as the decomposed two-qubit native gates at the device-level are the most error-prone gates on existing systems. Moreover, native gates exhibit temporal and spatial variation in their error-rates. Consequently, sub-optimal native gate selection reduces the success-rate of quantum programs. To minimize the impact of two-qubit gate errors on quantum programs, compilers must identify the optimal native gate combination for the CNOTs.

Prior works mitigates the impact of hardware errors by using (1) circuit decompositions with minimal gates [58, 100, 132]

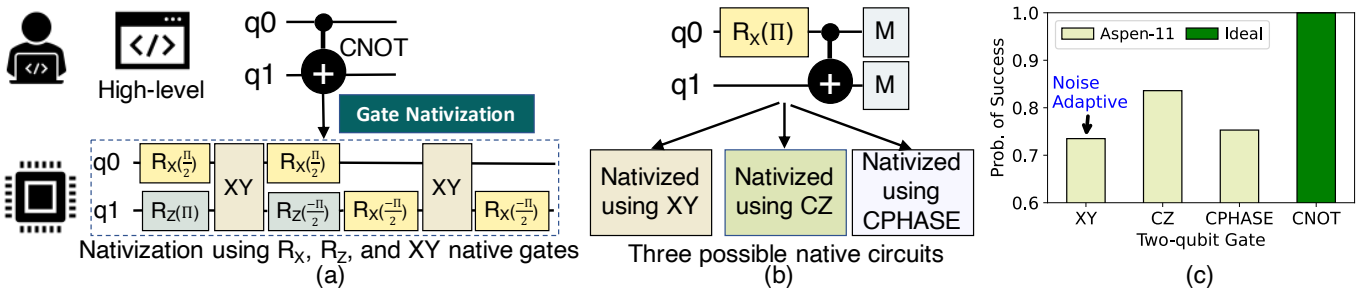


Fig. 1. (a) Gate nativization translates program instructions into native gates. (b) Rigetti devices allow three different two-qubit native gates (XY, CZ, and CPHASE). For the micro-benchmark shown here, we run three native circuits derived by decomposing the CNOT operation into XY, CZ, CPHASE native gates respectively. (c) When executed on Rigetti Aspen M-1, the CZ native circuit offers higher success-rate than the noise-adaptive circuit using the XY gate.

and (2) noise-adaptive optimizations that use the device error-model to steer more computations onto qubit devices and physical links with the lowest error-rates [72]. We can naturally extend noise-adaptive optimizations to nativize CNOTs on systems with multiple two-qubit native gates and select the native gate with the highest fidelity. However, our experiments on real devices show that noise-adaptive gate nativization is often sub-optimal at application-level, and the highest success-rate is obtained for a different combination of native gates.

This is because noise-adaptive compilers use the error-rates from device calibrations to maximize the success-rate of programs. This limits their performance to the accuracy of the calibration data which is imperfect in two ways. *Firstly*, the fidelity reported is obtained via randomized benchmarking [26, 54, 65] and only represents the average gate performance over all possible input quantum states. So, it may not accurately capture the error-trends of each native gate specific to the qubit state space of a given program. *Secondly*, the frequency of calibration varies depending on the native gate [4]. Also, device providers often use localized calibrations during which they may only calibrate some of the devices [53]. Thus, calibration data may not reflect the accurate error characteristics as device drifts are frequent in quantum systems [51, 115]. We explain the limitations of noise-adaptive nativization using a micro-benchmark, shown in Figure 1(b), that rotates the first qubit along the X-axis by π and uses it to control the second qubit. The correct output of this circuit is `11`. We prepare three circuits by nativizing the CNOT using XY, CZ, and CPHASE gates respectively and run them on Rigetti Aspen-11. Figure 1(c) shows that the noise-adaptive circuit using XY gate has a success-rate of 73%, whereas the circuit using CZ gate has a higher success-rate of 84%. Thus, there exists a gap between noise-adaptive native gate selection and the native gate that maximizes success-rate at application-level. This paper focuses on closing this gap.

In this paper, we propose *Application-specific Native Gate Selection (ANGEL)* – a software framework for efficient nativization of quantum programs. As the error-rates of two-qubit native gates are an order of magnitude higher than single-qubit gates, ANGEL focuses on nativization of CNOT operations. ANGEL maintains a list of all CNOT operations in a program, the (1) device links they will execute on, and (2) the native gate used to translate each of them. We refer to this as a *native gate sequence*. The goal of ANGEL is to learn the optimal sequence. The sequence that maximizes success-rate depends on the program and device characteristics. So, we cannot rely on generalized micro-benchmark circuits, as in Figure 1(b), to characterize the device and assist ANGEL in identifying the optimal sequence specific to a program. Moreover, capturing changes in device errors require frequent characterizations, increasing the overheads. Also, the characterization complexity scales exponential with the system size and is impractical.

ANGEL overcomes these limitations by learning the optimal native gate sequence independently for each program. However, this is non-trivial. Hypothetically, if the program output was known, ANGEL could evaluate the effectiveness of different native gate sequences using a trial-and-error approach and select the one that maximizes the success-rate. Unfortunately, the program output is unknown. To tackle this challenge, ANGEL leverages the insight that the optimal native gate sequence depends on the program and creates a CopyCat circuit that imitates the program structure but uses Clifford gates to replace the non-Clifford operations. The Clifford group comprises of X, Z, H, S, and CNOT gates and can be efficiently simulated on a conventional computer, unlike non-Clifford gates [54]. As CNOT is a Clifford gate, the CopyCat entirely imitates its usage from the input program. ANGEL (1) simulates the CopyCat on a conventional machine to obtain its noise-free output, (2) uses it to learn the native gate sequence that maximizes its success-rate on the NISQ device, and (3) uses this sequence to nativize the given program.

Although the CopyCat enables ANGEL to learn the optimal native sequence, the complexity of the trial-and-error method scales exponential with the number of CNOT operations in a program. To overcome this drawback, first, ANGEL uses a localized algorithm that initiates the search using the noise-adaptive native gate sequence as a *reference* point. This narrows down the search space. Second, ANGEL attempts to find a better candidate than the reference by altering the native gate for all operations on each link used by the program, one link at a time. Third, ANGEL continuously updates the reference each time an alternate native gate that increases the success-rate of the CopyCat is found for a link. The complexity of this search scales linear with the number of links used.

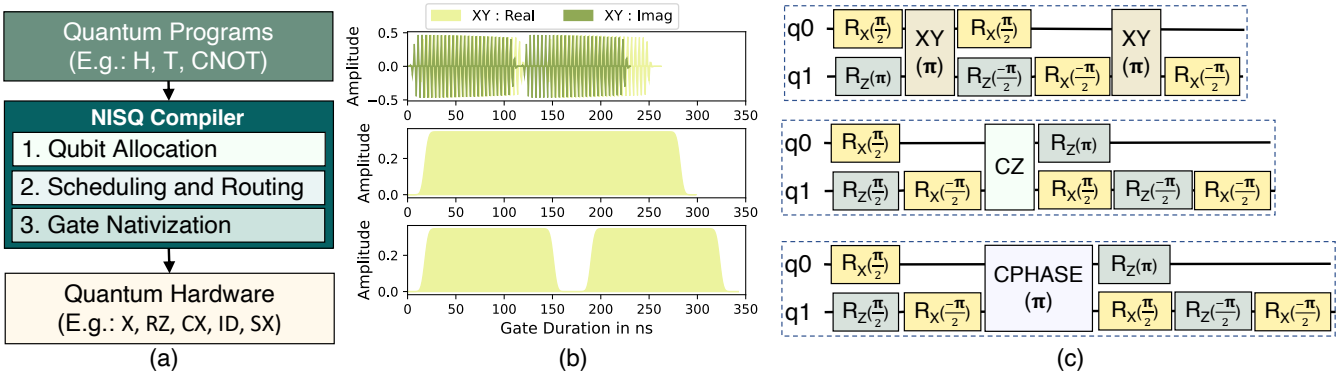


Fig. 2. (a) Compiler nativizes high-level instructions. (b) Rigetti machines support three 2-qubit native gates– XY, CZ, and CPHASE that differ in their pulses. (c) A CNOT gate can be nativized using any of them. Although the XY and CPHASE pulses are shorter, a CNOT using them requires two such pulses.

Our evaluations on Rigetti Aspen-11 show that ANGEL improves fidelity by 1.40x on average and by up-to 2x.

To that end, this paper makes the following contributions:

1) We show that naively adopting a noise-adaptive strategy for gate nativization is sub-optimal at the application-level.

2) We propose *Application-specific Native Gate Selection (ANGEL)*, a software framework to learn the optimal native gate for decomposing each two-qubit operation in a program.

3) ANGEL employs CopyCats and a localized search algorithm to identify the optimal native gate sequence with high accuracy in a scalable manner for practical adoption.

II. Background

A. Quantum Operations at Programming vs. Hardware Level

Quantum programs are usually written in high-level languages to allow seamless software development [17, 24, 96]. High-level languages also offer strong abstractions between the target operations in an algorithm and the underlying hardware on which the programs will be executed. On the other hand, the *native* or *basis* gates of a quantum computer refers to the set of low-level instructions closer to the hardware that can be directly executed on the device. Native gates are specific to each device. For example, the native gate set on IBMQ systems differ from that on Rigetti or Google machines. To run a quantum program, a compiler translates each high-level instruction of a program into an equivalent set of operations using the device-specific native gate set. For example, any quantum program must be translated by the compiler to a combination of gates from the set [*X, RZ, CX, ID, SX*] for execution on IBMQ systems [43], as shown in Figure 2(a).

B. Multiple Native Gates for Better Expressivity

Superconducting qubits, used in most existing quantum systems [2, 4, 43], are controlled by microwave pulses at the lowest level. Each native gate denotes a sequence of complex-valued analog pulses that can control the qubits to achieve the desired functionality. To entangle two qubits, the two-qubit native gate pulse tunes the frequencies of the interacting qubits to a desired operating point. These interaction frequencies can only be chosen from a limited range. As system sizes scale, the frequency space becomes crowded [55]. This increases the number of sources of unwanted couplings between qubits and lowers the device performance due to crosstalk [12, 89, 94]. Moreover, having only a single two-qubit native gate forces the compiler to express every program instruction using it. The limited expressivity often increases the total number of native gates in the quantum executable [56]. To alleviate the problem of frequency crowding and limited application expressivity, recent quantum systems support a richer native gate set with multiple two-qubit native gates. For example, Rigetti devices support three different two-qubit native gates, namely XY, CZ, and CPHASE. These native gates differ from each other in their pulse-level implementations, as shown in Figure 2(b).

Generally, most quantum computers can provide multiple native two-qubit gates. For example, while only the CR or ECR native gate is exposed to the software on current IBMQ systems, these machines are also capable of performing the RIP gate [45]. Similarly, trapped ion machines can perform multi-qubit Mølmer–Sørensen gates [121] as well as the CNOT gate. As gate fidelities improve and quantum systems scale, exposing multiple two-qubit native gates to the compiler provides opportunities for enhanced software optimizations. In practice, the only publicly accessible devices today with this capability are from Rigetti. In this work, we use Rigetti Aspen systems to demonstrate how the choice of multiple two-qubit gates can be leveraged to improve compilation results.

C. NISQ Compilation

NISQ compilation has three steps, shown in Figure 2(a): (1) qubit mapping that allocates a physical qubit to each program qubit, (2) scheduling and routing, and (3) gate nativization. During scheduling and routing, the compiler schedules operations if their dependencies are resolved and they can be directly executed. If a CNOT between non-adjacent physical qubits is found, the compiler adds SWAPs to route them next to each other. A SWAP generally comprises of three CNOTs that exchanges the state of two qubits and enables qubit relocation, although in specific cases, fewer CNOTs may be used [62]. During *gate nativization*, the scheduled and routed

operations are decomposed into native gates. For systems with multiple two-qubit native gates, a CNOT can be decomposed using either of them. For example, Figure 2(c) shows possible decompositions of a CNOT on Rigetti Aspen systems.

D. Problem and Motivation

Minimizing the impact of errors in CNOT operations is crucial to improve the success-rate of programs. Prior works have mainly focused on minimizing the total number of CNOT operations by reducing SWAPs [58, 132] and cancelling gates [67]. Advanced compilers leverage noise-adaptive optimizations to select SWAPs with the highest probabilities of success by using the gate fidelities from device calibrations [72]. However, these works do not consider the impact of gate nativization on application success-rate, especially on recent devices that offer multiple two-qubit native gates.

1) Impact of Gate Nativization at Application-Level: To study the impact of gate nativization on program success-rate, we execute a 5-qubit GHZ benchmark [34] on Rigetti Aspen-11. Ideally, the circuit produces '00000' and '11111' with 50% probabilities each. It uses four CNOTs and each of them can be decomposed into either of XY, CZ, or CPHASE gates. Thus, the total number of unique native gate combinations possible for this circuit is $3^4 = 81$. We run these 81 circuits on Aspen-11 and compute the success-rate (sum of the probabilities of the correct outcomes). Figure 3 shows the success-rate of the 81 circuits. We observe that the gate combination that maximizes success-rate at runtime ([*XY, CZ, CZ, CZ*]) differs from the combination obtained from noise-adaptive native gate selection ([*XY, XY, CZ, XY*])[1]. Noise-adaptive gate nativization uses the gate fidelities from device calibrations to select the native gate with the highest fidelity for each CNOT. The maximum success-rate for this GHZ benchmark is 3x the success-rate obtained using noise-adaptive gate nativization.

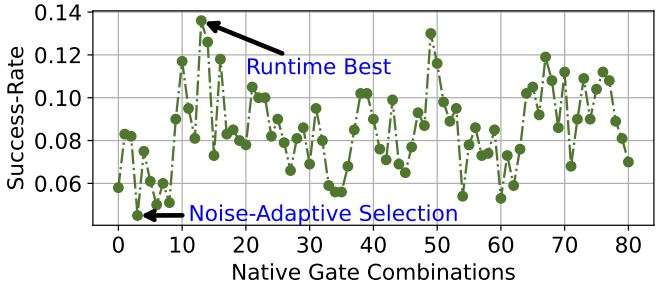


Fig. 3. Impact of gate nativization on the success-rate of a 5-qubit GHZ benchmark executed on the Rigetti Aspen-11 device.

Problem

Noise-adaptive gate nativization for CNOT operations is sub-optimal at application-level.

[1]The original noise-adaptive compiler [72] only focuses on qubit mapping and routing. This is orthogonal to the implementation here, where we naturally adopt a similar strategy for nativizing CNOT operations.

2) Why is Noise-Adaptive Gate Nativization Sub-optimal?: Noise-adaptive optimizations are limited by the accuracy of the calibration data. *Calibration* is the process during which the optimal control parameters for the on-chip quantum components are determined. Accurate calibration is essential for high fidelity quantum operations in quantum computers. Calibration uses *randomized* and *interleaved randomized benchmarking* to estimate the *average* gate fidelities by running circuits that implement long sequences of random gates chosen from the Clifford group [26, 54, 65]. However, this means that the fidelity data from calibration only represents an average and may not accurately assess the error-trends of a particular native gate in a specific program. Moreover, although device providers periodically benchmark their systems, the frequency depends upon the native gate. For example, CPHASE gates are benchmarked less frequently than others on Aspen-11 [4]. Device providers may also use frequent but localized calibrations of a small number of devices at-a-time [53]. As devices drift frequently, even the most recent calibration data may not capture the device error characteristics accurately. Relying on obsolete gate fidelity data may lead to sub-optimal native gate selection and thus, impact the success-rate of applications.

In this paper, *our **goal** is to design a compilation policy for efficient gate nativization*. To that end, we propose ***Application-specific Native Gate Selection (ANGEL)***.

III. Characterization Using Rigetti Aspen M-1

A. Fidelity Varies Depending on Quantum States

To understand the effectiveness of calibration data, we prepare the two micro-benchmark circuits shown in Figure 4. In the first circuit, we rotate qubit $q0$ by an angle θ about the X-axis and entangle it with qubit $q1$. The second circuit is similar to the first one, except that it rotates qubit $q0$ about the Y-axis. We create multiple such micro-benchmarks by controlling θ to prepare qubit $q0$ in different states. For our studies, we choose $\theta = [0, \frac{\pi}{3}, \frac{\pi}{2}, \frac{2\pi}{3}, \pi]$. For each circuit, we decompose the CNOT into XY, CZ, and CPHASE gates.

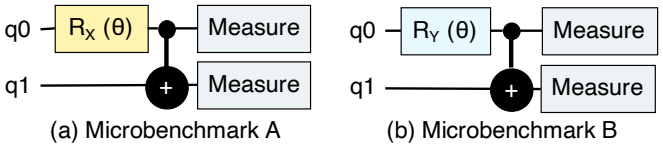
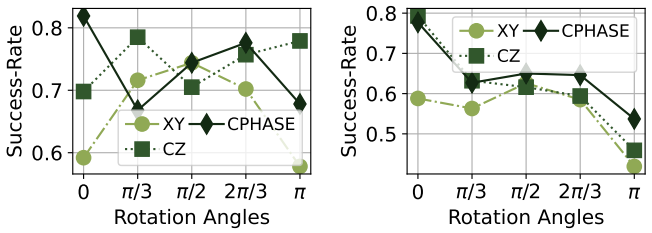


Fig. 4. Micro-benchmark circuits used for characterization

Simple characterization: Figure 5 shows the success-rate of these circuits when executed on a randomly chosen device link (110-117). As the circuits output a distribution, we compute success-rate using the total variation distance [120] between the noisy distribution obtained on the real device and that obtained on a simulator without noise. We observe that the native gate which maximizes the success-rate (1) depend on the state of the qubits, (2) device characteristics, and (3) may differ from the selection based on calibration data. For example, CZ gate is the noise-adaptive selection for this link, but XY and CPHASE gates outperforms in many cases.



(a) Circuit A on Link 110-117 (b) Circuit B on Link 110-117

Fig. 5. Success-Rate of micro-benchmarks on a random link of Aspen M-1

Extensive characterization: Next, we run each circuit on every connected pair of physical qubits (or links) on Aspen M-1. Thus, in total we run 5 (θ values) $\times L \times 3$ (native gates) $= 15N$ circuits, where L is the number of links. Although there are 103 physical links on Aspen M-1, the number of circuits executed (1460) is slightly lower than $15L = 1545$ because (1) few links do not support all three two-qubit native gates and (2) one of the links was disabled during our experiments. Figure 6 shows the success-rates of the circuits on Aspen M-1. Similar to previous experiments, we observe that with the exception of a few device links where a single native gate always outperforms the others (for example, CPHASE always outperforms on link number 44 between qubits 131 and 132), the native gate that maximizes the success-rate of the circuits show a strong dependence on the state of the qubits.

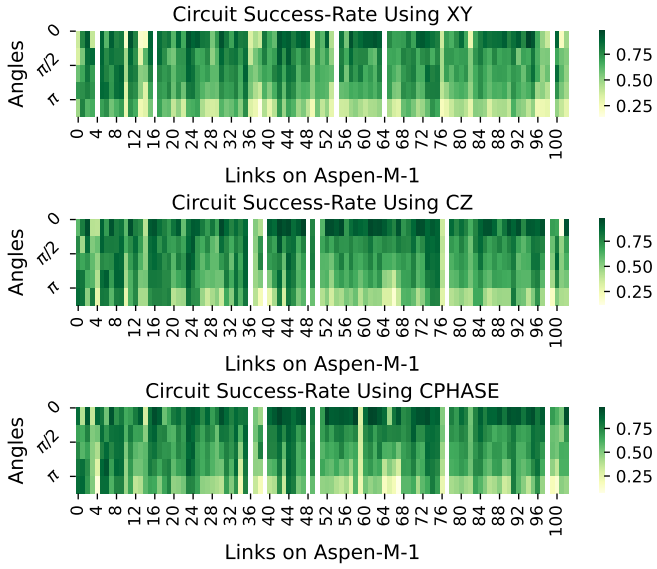
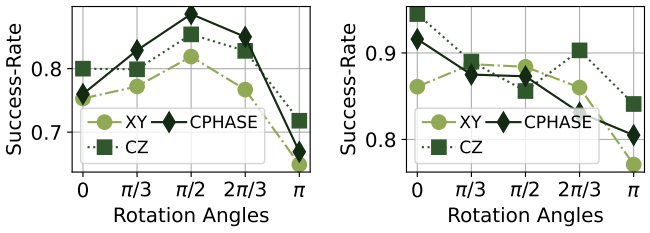


Fig. 6. Distribution of Success-Rate of micro-benchmark B across all links of Rigetti Aspen M-1.We observe similar trends using micro-benchmark A.

Observation 1

The native gate that maximizes success-rate of quantum circuits depends on the state of the qubits and error characteristics of the physical device, making it hard to generalize.

Characterization across calibrations cycles: We repeat these experiments across calibration cycles and observe that the trends in the success-rate of the circuits change between calibration cycles. For example, Figure 7 shows the success-rate of the micro-benchmark A on the physical link 100-107 across two calibration cycles. Hence, compilers cannot rely on such fine-grained generic characterization circuits to build their native gate selection policy as the characterization results turn obsolete quickly. Moreover, accurately capturing the error-trends require characterization circuits with operations on multiple qubits. This requires frequent and extensive characterization that scales exponential with the system-size.



(a) Calibration Cycle 1 (b) Calibration Cycle 2

Fig. 7. Success-Rate of micro-benchmark B on a randomly selected link (100-107) of Rigetti Aspen M-1 across two consecutive calibration cycles.

Observation 2

The trends in the success-rate of quantum circuits exhibit complex patterns across calibration cycles, making reliance on generalized micro-benchmarks impractical.

B. Calibration Data May Be Stale

We evaluate the historical calibration data for the Rigetti Aspen M-1 device and observe:

1) Device error-rates exhibit spatial and temporal variations, similar to other quantum computing systems [72]. For example, Figure 8 shows this variation in error-rates for a randomly chosen link (0-1) from the device.

2) However, we also observe that there is no temporal variation across several calibration cycles (shown by the plateaus in Figure 8). This is because some native gates are calibrated less frequently than others [4]. Moreover, some device providers may also locally calibrate only a subset of qubits each time [53]. Consequently, device drifts may go undetected.

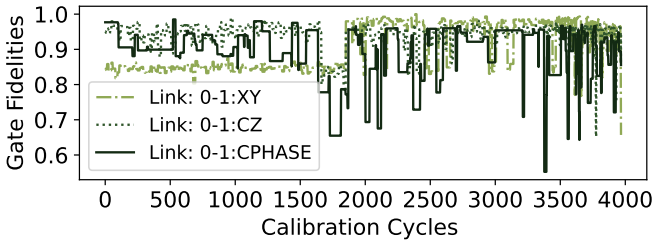


Fig. 8. Spatial and temporal variation of error-rates of three different two-qubit native gates on a randomly selected Rigetti Aspen M-1 link.

C. Need for Application-Specific Gate Nativization

Our studies show that the optimal combination of native gates which maximizes the success-rate depends on the program. For example, Figure 9 shows the optimal native gates for two 4-qubit benchmarks, namely GHZ_n4 and VQE_n4. Both these programs have three CNOT gates and thus, 27 possible native gate combinations. Even when we run them on the same physical qubits within the same calibration window, the native gate combinations that maximize the success-rate of the benchmarks are not identical owing to program-specific behavior. Also, the trends in success-rate for the different native gate combinations vary across the two benchmarks.

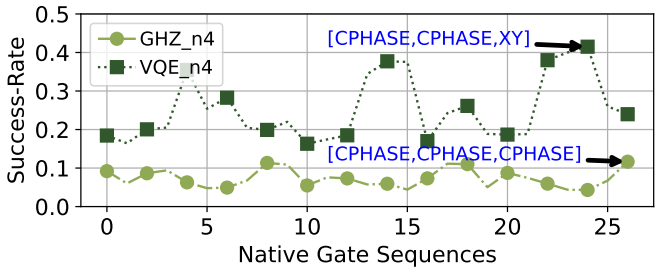


Fig. 9. Comparison of performance of native gate sequences.

Takeaway

Solely relying on calibration data or device characterization using generalized micro-benchmarks to perform gate nativization is typically inefficient. The success-rate can be improved by using application-specific gate nativization.

IV. ANGEL

Increasing the probability of successful execution of NISQ programs require compilers to perform efficient nativization of two-qubit CNOT operations. We propose *Application-specific Native Gate Selection (ANGEL)*, a software framework that identifies the combination of native gates for decomposing each CNOT operation in a program. We term any such combination as a *native gate sequence*. Next, we discuss an overview of ANGEL, the challenges in identifying the optimal native gate sequence, our insights, and design implementation.

A. Design Overview

Figure 10 shows an overview of ANGEL. To compile a program, its qubits are mapped, the instructions are scheduled, and SWAPs are added. ANGEL takes the scheduled and routed program and creates a *CopyCat*. Next, ANGEL uses a trial-and-error-method to learn the optimal native gate sequence for the CopyCat. Once identified, the same optimal native gate sequence is used to nativize the input program and generate its compiled circuit that can be executed on the NISQ device.

Impact on Compile Time: Qubit allocation and routing is the slowest step in compilation [91]. ANGEL only replaces the native gates in the scheduled and routed program to generate

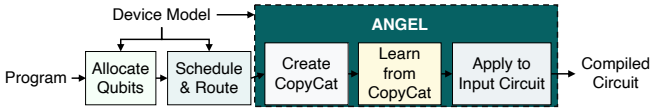


Fig. 10. Design overview of ANGEL.

different CopyCats as shown in Figure 10 and therefore, it does not increase the compilation time significantly.

B. Challenges in Optimal Native Gate Selection

The key challenges in identifying the optimal native gate sequence for a program are described next.

Cannot Rely on Device Characterization: Our experiments in Section III show that naively using calibration data for gate nativization is inefficient and may lead to sub-optimal success-rate during program execution. Also, compilers cannot use generalized micro-benchmarks, similar to the ones discussed in Section III, to overcome this drawback because:

1) The optimal sequence depends on the characteristics of the devices and application. Generic micro-benchmarks may not adequately capture the trends specific to the input circuit.

2) Generalized characterization will require more sophisticated micro-benchmarks to capture the complex error-trends when operations involve multiple qubits (similar to real programs) and more extensive studies, increasing the overheads.

3) To account for changing trends across calibration cycles, the micro-benchmarks must be frequently re-run, increasing the characterization overheads further in large systems.

All these factors combined make reliance on extensive device characterization for efficient gate nativization impractical.

Do Not Know the Output of the Given Program: The goal of ANGEL is to identify the native gate sequence that maximizes the success-rate of the input program. If the output of the program is known, ANGEL can test the efficacy of various sequences and select the one that maximizes its success-rate. Unfortunately, the program output is unknown and therefore, ANGEL cannot directly adopt this strategy.

Exponentially Large Search Space: Hypothetically, even if we assume that the correct output of the input program is available to ANGEL, identifying the optimal native gate sequence by using a naive trial-and-error method is impractical, as the search space scales exponentially. For example, each CNOT operation can be translated using three different two-qubit native gates on Rigetti Aspen devices and therefore, the total search space comprises of 3^N native gate sequences, where N is the total number of CNOT gates in the program.

C. Insight 1: Create CopyCats using Clifford Gates

The hardness of simulating most practical quantum circuits efficiently on conventional machines stems from the usage of non-Clifford gates. On the other hand, the output of circuits comprising of only Clifford operations can be easily determined by simulating them on classical systems [54]. We leverage this insight to design a *CopyCat*, which is a circuit that *imitates* the input program but uses only Clifford gates.

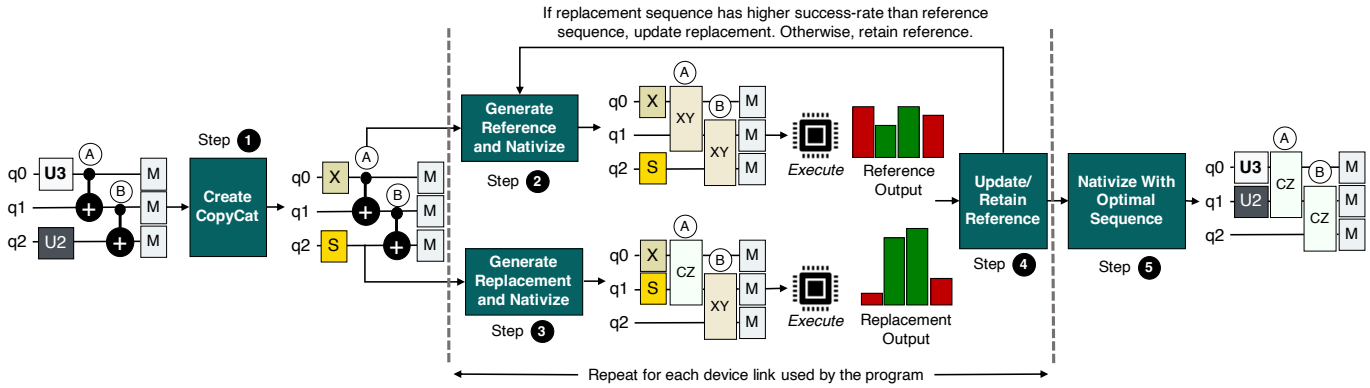


Fig. 11. Design Implementation of ANGEL: (1) It creates a CopyCat, (2) initializes a reference, and (3) generates replacement candidate sequences. It uses these sequences to nativize the CopyCat and executes them on the device. (4) If the success rate of the replacement native circuit is greater than the reference circuit, the reference sequence is updated to the reference sequence. ANGEL terminates the search once it has attempted to replace the native gates for each device link used by the program. (5) Finally, ANGEL uses the most up-to date reference sequence (optimal) to nativize the input circuit.

As the CopyCat imitates the input circuit structure and CNOT is a Clifford gate, the impact of different gate sequences on its success-rate is similar to input circuit. Thus, ANGEL uses the CopyCat to identify the optimal sequence by (1) applying different sequences and executing it on the device and (2) selecting the one that maximizes its success-rate. ANGEL nativizes the input program using the optimal sequence.

Although designing a CopyCat provides ANGEL a mechanism to identify the optimal native gate sequence, the exponentially complex search remains a major bottleneck.

D. Insight 2: Evaluate a Restricted Local Search Space

To reduce the search complexity, ANGEL only evaluates a limited number of native gate sequences that corresponds to a local region in the search space.

Starting with a Good Reference Sequence: ANGEL starts the search by selecting native gates with the highest fidelity from calibration data. Note that although the optimal sequence will likely use a different combination of native gates than this reference, the latter sequence still enables ANGEL to *initiate* the search from a good *reference* point and the best-known candidate prior to the search. ANGEL searches for the optimal sequence by using a trial-and-error method to replace the native gates in the reference sequence, thereby restricting the search space explored by ANGEL.

Learn Locally and Adapt Frequently: ANGEL uses two key features to further reduce the search complexity: *mass replacement* and *continuous update*. Instead of learning for each CNOT individually, ANGEL searches for a single native gate replacement for all CNOT operations on a particular link that maximizes the success-rate of the CopyCat (more on this discussed in the next subsection). We refer to this as mass replacement. Moreover, if a replacement is found that offers higher success-rate for the CopyCat, the reference sequence is updated before proceeding further. This property is called continuous update. It enables ANGEL to account for the errors from the replacement native gate candidate in future searches.

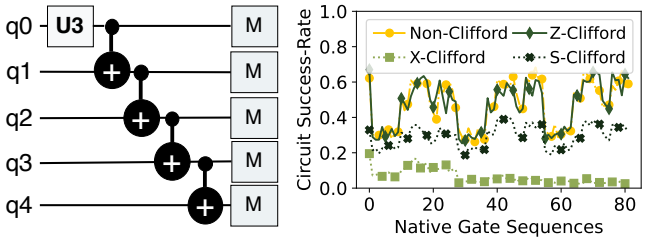
E. Design Implementation

ANGEL is integrated in the gate nativization phase of compilation when the high-level operations of the scheduled and routed circuit are decomposed using native gates. Next, we discuss the key features of the implementation of ANGEL.

1) Step-1 → Designing CopyCat: ANGEL generates a CopyCat of the input program, as shown in Figure 11. To create a CopyCat, ANGEL replaces the Non-Clifford gates of the input circuit using gates from the Clifford group that comprises of [X, Y, Z, H, S, $CNOT$] gates. The goal of the CopyCat is to *imitate* or *copy* the input program while ensuring that that it can be efficiently simulated.

Clifford replacements Must Be Carefully Selected: To ensure that the CopyCat can effectively imitate the input program, the Clifford replacements must be accurately chosen. Failure to do so results in a poor CopyCat that cannot capture the success-rate trends of the input circuit accurately. For example, we take the Figure 12(a) circuit and prepare three CopyCats- each using X, Z, and S to replace the U3 gate. The U3 gate is a generic single-qubit rotation gate whose angles are chosen randomly. We nativize each CopyCat using all possible native gate sequences. As there are 4 CNOTs, there are $3^4 = 81$ possibilities. We run the circuits on Rigetti Aspen-11 and compute their success-rates, shown in Figure 12(b). We observe that replacing non-Clifford operations with Clifford gates enable us to learn the fidelity trends of the input circuit, only if is replaced accurately. For example, the Z and S CopyCats have high correlation with the input circuit, with Spearman's Correlation Coefficients (SCCs) 0.87 and 0.89 respectively, whereas the X CopyCat shows poor correlation and its SCC is only 0.13. Note that SCC is a statistical measure to compute the strength of association or relationship between two variables [123]. A higher SCC denotes greater correlation or a stronger relationship between the data.

To obtain the Clifford equivalent of a non-Clifford operation, ANGEL computes the operator norm between the unitaries. The operator norm [122] computes the distance between



(a) Micro-benchmark (b) Success-Rate of Circuits

Fig. 12. (a) A non-Clifford that prepares qubit $q0$ in an arbitrary state using the generic single-qubit rotation U3 gate and performs a sequence of CNOTs. (b) The success-rate of the Non-Clifford circuits and its three CopyCats.

two unitaries U and V, as described by Equation 1. The closest Clifford equivalent is the one whose unitary is at the minimum distance from the target non-Clifford operation to be replaced. To increase the sensitivity of the CopyCat, ANGEL does not utilize the H as it creates an equal superposition state.

$$\|U - V\|_\infty := \max_{|\psi\rangle \neq 0} \frac{\|(U - V)|\psi\rangle\|_2}{\||\psi\rangle\|_2} \quad (1)$$

Using only Clifford gates can make identifying the optimal sequence difficult. This is because the CopyCat in that case produces an output distribution with high entropy, making it insensitive to changes in native gate selection. We observe this behavior in our experiments and similar trends are reported in prior studies [20]. To create a CopyCat that can imitate the input program more accurately, we use a limited number of non-Clifford gates in the initial layer of the circuit and replace all other non-Clifford gates with Clifford equivalents. We limit the number of non-Cliffords to less than 20 to keep the simulation complexity tractable, based on prior studies [19, 20]. Figure 13 shows an example circuit and its CopyCats. The usage of a limited number of non-Clifford gates in the CopyCat is inspired from a recent work on enabling effective dynamical decoupling to reduce idling errors [20].

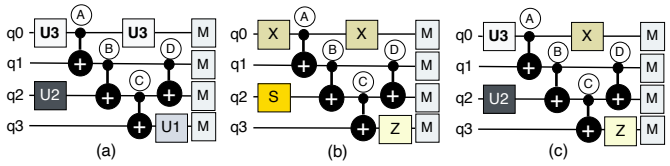


Fig. 13. (a) A circuit, its (b) Clifford-only CopyCat, and (c) CopyCat.

2) Step-2 → Initialize Reference Sequence: At any point in the search for the optimal native gate sequence, ANGEL maintains a *reference* sequence, which represents the native gate combination that offers the highest success-rate for the CopyCat at that point of time. The reference sequence is initialized at the beginning and is continuously updated as the search progresses. At the end of the search, the reference sequence denotes the optimal sequence learned by ANGEL.

ANGEL requires a strong candidate sequence to initiate the search and this step is very crucial as future candidates in the search are derived from it. Failure to select a high quality reference during initialization restricts ANGEL from quickly identifying the region with the optimal sequence in the search space and may require a large number of CopyCats, increasing the overhead of ANGEL. Therefore, to initialize the reference to a strong candidate, ANGEL selects the native gate for each CNOT as the one with the maximum fidelity for the link it will be run on. This is inspired from prior noise-adaptive compilers [72] and this noise-adaptive native gate sequence is the best-known sequence prior to the search done by ANGEL.

For example, Figure 14 (a) shows a circuit mapped to (b) a NISQ device. The figure also shows the (1) CNOT gates that will be executed and (2) the highest fidelity native gate (obtained from calibration) for each link. ANGEL starts with `Reference` = [Ⓐ→ {0-1: XY}, Ⓑ→ {1-2: CZ}, Ⓒ→ {2-3: CPHASE}, Ⓓ→ {0-1: CZ}]. The CopyCat is nativized using this sequence and executed. The success-rate of this CopyCat is used to evaluate future replacement candidates.

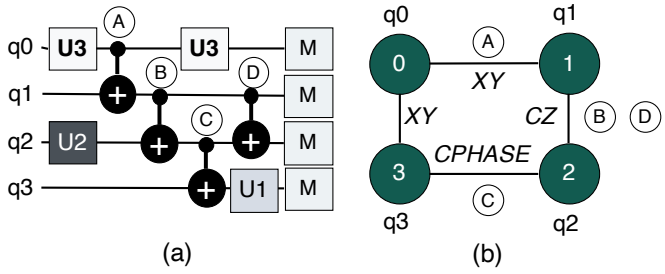


Fig. 14. (a) A circuit mapped to (b) a quantum device. The labels adjacent to each link shows the program CNOTs mapped to them and the native gate with the highest fidelity for that particular link.

3) Step-3 → Generate Replacement Sequence Candidates: Next, ANGEL generates *replacement* sequence candidates by scanning through the operations that occur on each link of the circuit. For a selected link, it generates a replacement sequence by altering the native gates for all the operations on that link. ANGEL makes this trade-off here to reduce the complexity of the search. In the example above, ANGEL creates two replacement candidates by replacing XY gate for CNOT Ⓐ on link 0-1 with CZ and CPHASE gates. Thus, the replacements are `Replacement-A` = [Ⓐ→ {**0-1: CZ**}, Ⓑ→ {1-2: CZ}, Ⓒ→ {2-3: CPHASE}, Ⓓ→ {0-1: CZ}] and `Replacement-B` = [Ⓐ→ {**0-1: CPHASE**}, Ⓑ→ {1-2: CZ}, Ⓒ→ {2-3: CPHASE}, Ⓓ→ {0-1: CZ}]. To reduce the overheads, ANGEL performs *mass replacement* during which the native gates are altered for any operation on the target link. For instance, while generating replacement candidates for link 1-2, the native gates for both CNOTs Ⓑ and Ⓓ are altered. By default, ANGEL uses the program order to generate replacement candidates and keeps the design simple.

4) Step-4 → Continuous Update of Reference Sequence: ANGEL (a) creates two CopyCats using the replacement candidates, and (b) runs them on the NISQ device. If a replacement CopyCat offers greater fidelity than the reference Copy-

Cat, the reference is updated to the corresponding replacement sequence. For example, if `Replacement-A` CopyCat has higher fidelity, the reference is updated to `Replacement-A`. We call this feature *Continuous Update*. It allows us to quickly adapt the search space of ANGEL and evaluate the impact of errors from each replacement on future replacements.

ANGEL repeats Steps (3) and (4) for each link at-a-time, until the algorithm has attempted replacements for each device link used by the program. We illustrate this using the example in Figure 15. We observe that ANGEL uses 8 CopyCats and identifies `Replacement-E` = [Ⓐ→ {**0-1: CZ**}, Ⓑ→ {1-2: CZ}, Ⓒ→ {**2-3: XY**}, Ⓓ→ {0-1: CZ}] as the optimal sequence. Note the simultaneous or mass replacement of native gates for CNOTs Ⓑ and Ⓓ in Figure 15.

CNOT	Link	Native
Ⓐ	0-1	XY
Ⓑ	1-2	CZ
Ⓒ	2-3	CPHASE
Ⓓ	1-2	CZ

Reference

CNOT	Link	Native
Ⓐ	**0-1**	**CZ**
Ⓑ	1-2	CZ
Ⓒ	2-3	CPHASE
Ⓓ	1-2	CZ

Replacement-A

CNOT	Link	Native
Ⓐ	**0-1**	**CPHASE**
Ⓑ	1-2	CZ
Ⓒ	2-3	CPHASE
Ⓓ	1-2	CZ

Replacement-B

(a) Link 0-1: Replacement-A outperforms, update reference.

CNOT	Link	Native
Ⓐ	0-1	CZ
Ⓑ	1-2	CZ
Ⓒ	2-3	CPHASE
Ⓓ	1-2	CZ

Reference

CNOT	Link	Native
Ⓐ	0-1	CZ
Ⓑ	**1-2**	**XY**
Ⓒ	2-3	CPHASE
Ⓓ	**1-2**	**XY**

Replacement-C

CNOT	Link	Native
Ⓐ	0-1	CZ
Ⓑ	**1-2**	**CPHASE**
Ⓒ	2-3	CPHASE
Ⓓ	**1-2**	**CPHASE**

Replacement-D

(b) Link 1-2: Reference outperforms, retain it.

CNOT	Link	Native
Ⓐ	0-1	CZ
Ⓑ	1-2	CZ
Ⓒ	2-3	CPHASE
Ⓓ	1-2	CZ

Reference

CNOT	Link	Native
Ⓐ	0-1	CZ
Ⓑ	1-2	CZ
Ⓒ	**2-3**	**XY**
Ⓓ	1-2	CZ

Replacement-E

CNOT	Link	Native
Ⓐ	0-1	CZ
Ⓑ	1-2	CZ
Ⓒ	**2-3**	**CZ**
Ⓓ	1-2	CZ

Replacement-F

(c) Link 2-3: Replacement-E outperforms, update reference.

Fig. 15. ANGEL steps. Note continuous update of the reference and mass replacement for link 1-2 (figure is for the purpose of illustration only).

5) Step-5 → Transfer Learning from CopyCats to Program: Finally, ANGEL uses the optimal native gate sequence (the most up-to-date reference sequence) to nativize the input program and execute it on the NISQ hardware. For example, Figure 16 shows how ANGEL starts with a noise-adaptive sequence and eventually migrates to a more accurate one for the circuit shown in Figure 14(a). Note that most of the time in compilation is spent on qubit routing [91] and gate nativization only replaces each high-level operation with the corresponding native gate implementation. As CopyCats only evaluates different native gate sequences, it does so on the same scheduled and routed program and therefore, CopyCats do not increase the compilation overheads significantly.

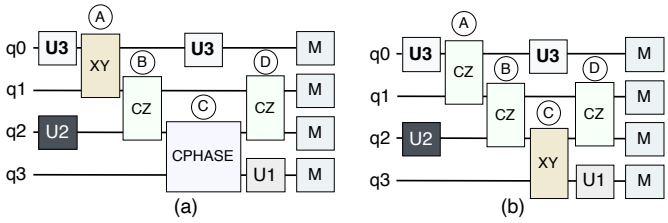


Fig. 16. Compiled circuit using (a) noise-adaptive native gate selection and (b) ANGEL's native gate selection (final schedule of the input program).

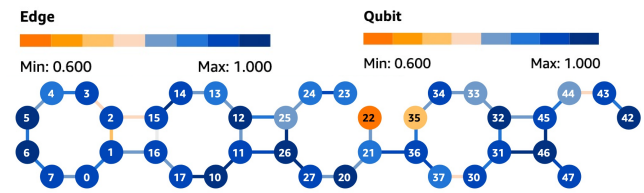
V. Methodology

A. Quantum Hardware

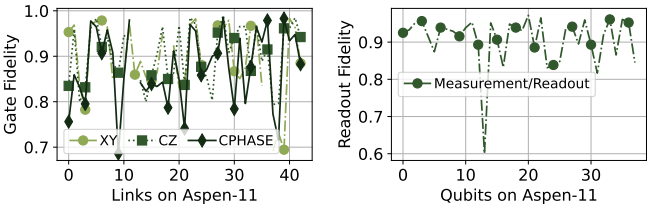
For Characterization: We use 38-qubit Rigetti `Aspen-11` and 80-qubit Rigetti `Aspen-M-1` for our characterization studies in Section III. We access them via Amazon Braket [4].

For Evaluating ANGEL: We use 38-qubit Rigetti `Aspen-11` for our final evaluations in Section VI. Unfortunately, the `Aspen-M-1` device was inaccessible due to maintenance operations when we ran our final evaluation experiments (presented in Section VI).

The Rigetti Aspen devices are the only publicly available quantum computers that support multiple two-qubit native gates. Figure 17 shows an overview of `Aspen-11`. The qubits and edges are color-coded based on their readout error-rates and CPHASE error-rates respectively.



(a) Device topology



(b) Gate fidelities (c) Readout fidelities

Fig. 17. (a) Device topology, fidelity of (b) two-qubit native gates, and (c) measurement/readout operations on Rigetti Aspen-11.

Note that although IBMQ systems are publicly available and used for several compiler-level studies, they only expose one two-qubit native gate, the CX gate, to the compiler. So, we cannot perform experiments on these machines.

B. Baseline and Competing Native Gate Selection Policies

We compare the performance using the following policies:

1) **Baseline (Noise-Adaptive)**: For the baseline, we adopt a noise-adaptive native gate selection strategy.

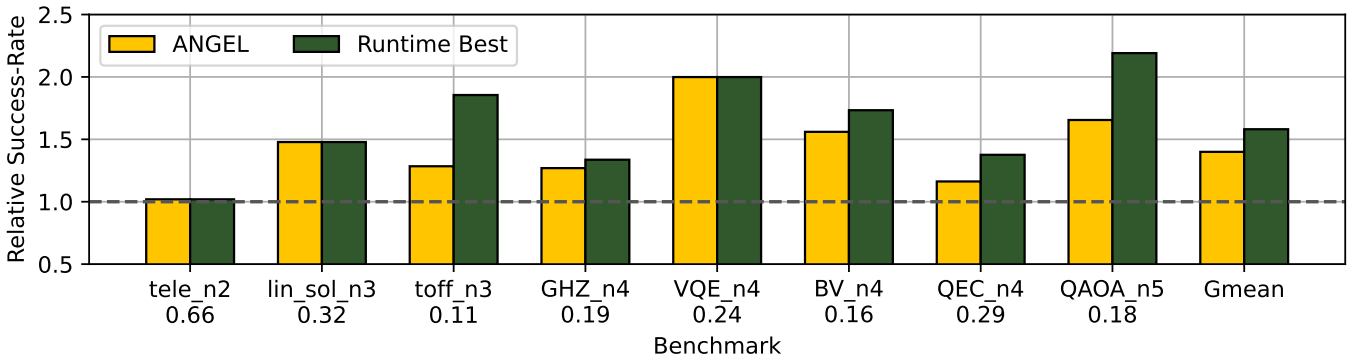


Fig. 18. Program Success-Rate (SR) relative to noise-adaptive native gate selection. The label below each benchmark specifies the SR of the baseline.

2) **ANGEL**: We execute the program using the optimal native gate sequence predicted by ANGEL. This corresponds to the best sequence using ANGEL's localized search algorithm.

3) **Runtime Best**: We run the program using all possible sequences and select the one that maximizes fidelity at runtime. There is a total of 3^N possibilities for a program with N CNOTs. This enables us to evaluate the effectiveness of the CopyCats and assess the overall performance of ANGEL.

C. Benchmarks

Table I shows an overview of the benchmarks used in our evaluations. These benchmarks are derived from several prior works on software error-mitigation for NISQ devices [57, 58, 72, 102, 109, 114, 124, 130, 132]. The measured quantum volume of Rigetti Aspen-11 device ranges between 2 to 4 [87] which means that square circuits of up to size 4 can be run reliably. Therefore, we select the size of the benchmarks around this range for our evaluations in Section VI.

TABLE I
SUMMARY OF BENCHMARKS USED

Description	Name	Qubits	CNOTs
Teleportation	tele_n2) [57]	2	2
Linear Solver	lin_sol_n3 [57]	3	3
Toffoli Gate	(toff_n3) [57]	3	9
Greenberger-Horne -Zeilinger [34]	GHZ_n4	4	3
Variational Quantum Eigensolver [69]	VQE_n4 [88]	4	3
Bernstein Vazirani	BV_n4 [11]	4	6
Quantum Error Correction	QEC_n4 [16]	4	5
Quantum Approximate Optimization Algorithm [27]	QAOA_n5 [20]	5	4

D. Figure-of-Merit

We measure the *Success-Rate (SR)* of a program by using the Total Variation Distance [120] between the noise-free output distribution obtained on a simulator (P) and the noisy distribution obtained on the real device (Q), as shown in Equation (2). This metric is derived from various prior works [6, 20, 22, 83, 97]. Success-Rate is a bounded metric and ranges between 0 and 1, where 1 represents completely identical distributions. *Therefore, a higher Success-Rate is desirable while executing NISQ programs.*

$$\text{Success-Rate (SR)}(P,Q) = 1 - \sum_{i=1}^{k} || P_i - Q_i || \quad (2)$$

VI. Final Evaluations

A. Impact On Success-Rate of Applications

Figure 18 shows the Success-Rate (SR) of the benchmarks relative to a noise-adaptive native gate selection for different nativization policies. ANGEL improves the SR by 1.40x on average and by up-to 2x compared to the baseline. Figure 18 also shows the relative SR for the native gate sequence that maximizes the SR at runtime. We observe that ANGEL successfully closes the gap between noise-adaptive native gate sequence and the one that outperforms at runtime.

B. Understanding the Scalability of ANGEL

Rigetti devices allow three two-qubit native gates. So, the number of CopyCats required for an exhaustive search is 3^N, where N is the number of CNOTs in a program (post SWAP insertion). ANGEL's localized search algorithm reduces this complexity. ANGEL creates two replacement candidates for each device link. Instead of an exhaustive search, ANGEL learns the optimal native gate for one link at a time. Thus, ANGEL requires 1 reference CopyCat and $2L$ replacement CopyCats, where L is the number of links used by the program. For example, the `toff_n3` benchmark requires 9 CNOTs on 2 links (post SWAP insertion) on Aspen-11. An exhaustive search requires 19.7K CopyCats, whereas ANGEL requires only 5 CopyCats in total. Note that `toff_n3` benchmark data presented in Figure 18 uses the same native gate to decompose the SWAP (as the CNOTs are anyway on the same link). We do this to reduce the runtime of our experiment by limiting the search space from 19.7K circuits to 729 circuits. Table II compares the number of CopyCats required for ANGEL with respect to an exhaustive search for the circuits used in our evaluations.

TABLE II
NUMBER OF COPYCATS REQUIRED

Benchmark	Exhaustive Search	ANGEL
tele_n2	4	4
lin_sol_n3	81	5
toff_n3	19.7K	5
GHZ_n4	27	7
VQE_n4	27	7
BV_n4	729	7
QEC_n4	243	9
QAOA_n5	81	9

C. Understanding CopyCats at Application-level

Figure 19 shows Success-Rate of the `linearsolver_n3` benchmark and its CopyCats for different native gate sequences. The benchmark consists of 4 CNOTs and therefore, there are 81 possible sequences. We observe that the CopyCat can effectively imitate the actual program and its Success-Rate follows similar trends as that of the input program. The strong correlation enables CopyCats to accurately learn the optimal native gate sequence for the input program.

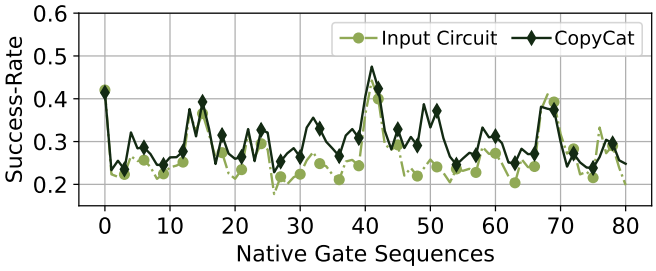


Fig. 19. Comparison of Success-Rate of a program and its CopyCat.

D. Impact of Reference Selection

Figure 20 shows that the improvement in SR is higher for the default ANGEL design using noise-adaptive sequence as a reference as compared to a search using a random reference.

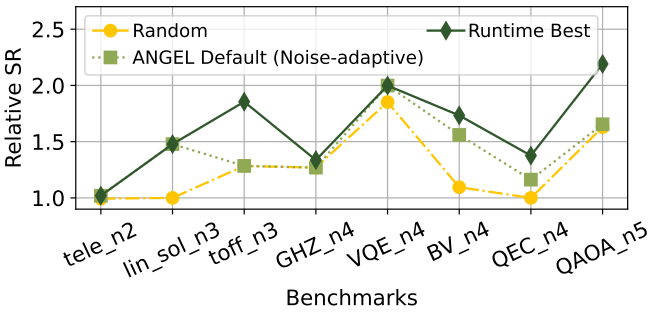


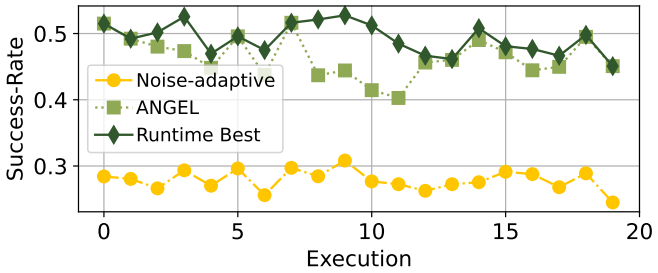
Fig. 20. Performance of ANGEL using a random vs. noise-adaptive reference.

E. Limitations of ANGEL

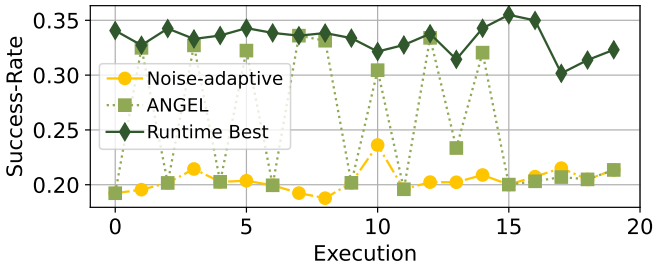
ANGEL has the following two limitations:

1) It cannot always *fully* close the gap between noise-adaptive and the runtime best sequence. This is due to (a) imperfections in the CopyCats and (b) the limited search space of ANGEL.

2) ANGEL, or any compiler, cannot alleviate strong device drifts. Device drifts may impact the learning step of ANGEL, causing it to perform poorly. For example, Figure 21 shows the Success-Rate (SR) of the GHZ_n4 benchmark when it is repeatedly executed within the same calibration window two times a day. We observe that device drifts cause variation in SR even for the same native gate sequence (noise-adaptive in this case). Also, they cause the native gate sequence that maximizes SR at runtime for each execution to vary. In the first example, ANGEL consistently outperforms the baseline, whereas in rare cases (as in second example), the efficacy of ANGEL reduces to almost the baseline.



(a) Repeated Executions (Example 1)



(b) Repeated Executions (Example 2)

Fig. 21. Success-Rate of a GHZ_n4 benchmark from repeated executions within the same calibration window.

We investigate the performance of the runtime-best sequences and observe that sequences [CPHASE, XY, CPHASE], [CZ, XY, CPHASE], [XY, XY, CPHASE] offers the maximum SR for 12, 5, and 3 out of the 20 iterations respectively in the first example (Figure 21(a)). Moreover, the sequence [CPHASE, XY, CPHASE] outperforms for several consecutive iterations, as shown in Figure 22. On the other hand, sequences [CPHASE, XY, CPHASE], [CZ, XY, CPHASE], [XY, XY, CPHASE] offers the maximum SR for 7, 7, and 6 out of the 20 iterations respectively in the second example (Figure 21(b)). This means that the sequence [CPHASE, XY, CPHASE] is significantly more stable during the first execution compared to the second execution which enables ANGEL to learn it more effectively. Note that the noise-adaptive sequence here is [CPHASE, XY, XY].

Device drifts are still not completely understood and hard to eliminate fully at the device level [51, 115]. Many prior works have demonstrated these fluctuations on Rigetti devices [31, 37, 98]. The Rigetti Aspen devices are some of the earliest machines that support multiple two-qubit native

gates and therefore, comparatively more error-prone than other devices (such as the ones from IBM). As these systems mature, we expect the impact of device drifts to reduce in the future.

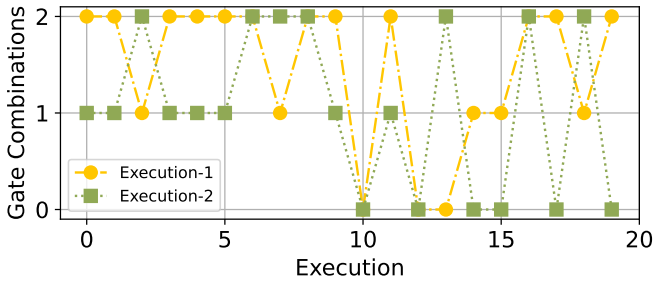


Fig. 22. Distribution of the runtime-best sequence for two executions of a GHZ_n4 benchmark discussed in Figure 21. Here, the gate combinations 0, 1, and 2 corresponds to the sequences [XY, XY, CPHASE], [CZ, XY, CPHASE], and [CPHASE, XY, CPHASE] respectively.

VII. Related Work

Designing a quantum computer requires full-stack solutions and interdisciplinary research [13, 66]. This has led to developments in managing quantum hardware in the cloud [21, 23, 70, 91, 93, 107], programming languages [10, 15, 18, 32, 33, 52, 68, 86, 104, 105, 108, 118], compilers [29, 36, 41, 58, 72, 113, 132], tool-chains to aid software development [38–40], micro-architecture [8, 25, 28, 49, 112], and quantum devices. As this paper focuses on the design of a compiler that improves the success-rate of NISQ application, we discuss related works on software error-mitigation and compare as appropriate.

Circuit Decomposition, Qubit Mapping and Routing Several recent compiler optimizations perform (1) application-specific error-mitigation [3, 59, 92, 126], (2) noise-aware circuit design [101, 116, 117], (3) circuit partitioning [110], (4) enhanced circuit synthesis and gate cancellations [67, 84, 85, 89, 99, 119, 128, 129], and (5) efficient qubit mapping and routing [7, 60, 62, 63, 76, 77, 80, 103, 125]. These approaches reduce gate counts, steer more computations to devices with lowest error-rates, and enable robust scheduling [72–75, 114]. ANGEL is orthogonal to such policies and uses the scheduled and routed program to perform efficient gate nativization.

Pulse Optimizations focus on optimal pulse control [30] and instruction aggregation to reduce program durations [100].

Software Post-processing schemes alter the noisy outputs by detecting errors [61, 131] or mitigating the impact of specific errors [22, 42]. While some of these function standalone [106, 111], others need additional data [22, 42, 81, 82].

A. Comparison with ADAPT

ADAPT is a software framework for efficient usage of dynamical decoupling (DD) sequences to reduce idling errors [20]. We specifically compare ANGEL with this work because the usage of CopyCats is inspired from ADAPT. ADAPT learns to enable or disable DD for each idle qubit in a program and sits outside the compiler tool-chain as idle time periods can only be determined after a circuit is nativized. On the contrary, ANGEL is integrated in the gate nativization step of compilation and reduces two-qubit errors. The search algorithm for the two are also different. ADAPT uses a simple enable/disable DD mechanism for small blocks of qubits. On the other hand, ANGEL starts its search from a good reference point and relies on continuous updates and mass replacements.

B. Comparison with Clifford Data Regression

Clifford data regression (CDR) [19] trains a model using many Clifford circuits with known outcomes that learns from the correlation between the expected outcome of the circuit and the noisy outcome from the real hardware. Then, it post-processes the output distributions of any program based on the predictions from the model. Unfortunately, CDR requires very large training datasets crafted from generic Clifford circuits, frequent re-training to adapt to the changing error characteristics of the devices, and post-processing overheads. ANGEL avoids these shortcomings and only relies on a fine-grained Clifford circuit that best mimics the characteristics of a given program to learn the optimal native gate sequence. Note that broadly, CDR and ANGEL target fundamentally different problems- while CDR post-process the noisy output distributions using software, ANGEL focuses on improving the gate nativization step specifically. Thus, we expect ANGEL can further improve the effectiveness of CDR by building better native gate implementations of the training Clifford circuits as well as by performing more accurate nativization of the input program. We reserve this for future work.

VIII. Conclusion

Gate nativization in quantum compilation translates high-level program instructions into low-level native gates. Our experiments show that noise-adaptive gate nativization is often sub-optimal at application-level for quantum systems that support multiple two-qubit native gates. This paper proposes *Application-specific Native Gate Selection (ANGEL)*, a software framework for efficient gate nativization. As the optimal native gate sequence depends on device and application characteristics, ANGEL creates a CopyCat that imitates a given program but can be simulated effectively on a conventional computer. Next, ANGEL uses this CopyCat to learn the optimal native gate sequence that maximizes its success-rate on the NISQ hardware. Finally, ANGEL uses this sequence to nativize the given program. To reduce the search complexity, ANGEL starts the search from a good reference point and uses a localized algorithm. Our experiments on Rigetti Aspen-11 shows that ANGEL improves the success-rate of applications by 1.40x on average and by up-to 2x compared to the baseline.

Acknowledgements

This work was done when Poulami Das was an intern in the Amazon Braket Science group. We thank Moin Qureshi for his support to pursue this work. We thank for Eric Peterson for his comments on the paper and feedback. We also thank Mao Lin, Jeffrey Hickey, and Jon Best for their help in the experimental setup. Last but not the least, we thank Richard Moulds and Simone Severini for their support and general guidance.

REFERENCES

[1] R. Acharya, I. Aleiner, R. Allen, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. Atalaya, R. Babbush *et al.*, "Suppressing quantum errors by scaling a surface code logical qubit," *arXiv preprint arXiv:2207.06431*, 2022.

[2] G. Q. AI, "Quantum computer datasheet," Accessed: June 19, 2021, https://quantumai.google/hardware/datasheet/weber.pdf.

[3] M. Alam, A. Ash-Saki, and S. Ghosh, "Circuit compilation methodologies for quantum approximate optimization algorithm," in *MICRO*. IEEE, 2020, pp. 215–228.

[4] Amazon, "Rigetti Superconducting Quantum Processors," https://aws.amazon.com/braket/quantum-computers/rigetti/, 2021.

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

[6] A. Ash-Saki, M. Alam, and S. Ghosh, "Experimental characterization, modeling, and analysis of crosstalk in a quantum computer," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–6, 2020.

[7] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong, "Time-sliced quantum circuit partitioning for modular architectures," in *Computing Frontiers*, 2020, pp. 98–107.

[8] J. C. Bardin, E. Jeffrey, E. Lucero, T. Huang, O. Naaman, R. Barends, T. White, M. Giustina, D. Sank, P. Roushan *et al.*, "29.1 a 28nm bulk-cmos 4-to-8ghz¡ 2mw cryogenic pulse modulator for scalable quantum computing," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 456–458.

[9] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[10] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, C. Blank, K. McKiernan, and N. Killoran, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018, https://pennylane.readthedocs.io/en/latest/.

[11] E. Bernstein and U. Vazirani, "Quantum complexity theory," *SIAM Journal on computing*, vol. 26, no. 5, pp. 1411–1473, 1997.

[12] S. Caldwell, N. Didier, C. Ryan, E. Sete, A. Hudson, P. Karalekas, R. Manenti, M. da Silva, R. Sinclair, E. Acala *et al.*, "Parametrically activated entangling gates using transmon qubits," *Physical Review Applied*, vol. 10, no. 3, p. 034050, 2018.

[13] F. T. Chong, D. Franklin, and M. Martonosi, "Programming languages and compiler design for realistic quantum hardware," *Nature*, 2017.

[14] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, pp. 4091–4094, May 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.74.4091

[15] R. Computing, "pyquil," Accessed: November 19, 2019, http://docs.rigetti.com/en/stable/.

[16] A. D. Córcoles, E. Magesan, S. J. Srinivasan, A. W. Cross, M. Steffen, J. M. Gambetta, and J. M. Chow, "Demonstration of a quantum error detection code using a square lattice of four superconducting qubits," *Nature communications*, vol. 6, no. 1, pp. 1–10, 2015.

[17] A. Cross, A. Javadi-Abhari, T. Alexander, N. de Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, P. Sivarajah, J. Smolin, J. M. Gambetta *et al.*, "Openqasm 3: A broader and deeper quantum assembly language," *ACM Transactions on Quantum Computing*, 2021.

[18] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," *arXiv preprint arXiv:1707.03429*, 2017, https://www.quantum-inspire.com/kbase/qasm/.

[19] P. Czarnik, A. Arrasmith, P. J. Coles, and L. Cincio, "Error mitigation with clifford quantum-circuit data," *Quantum*, vol. 5, p. 592, 2021.

[20] P. Das, S. Dangwal, S. S. Tannu, and M. Qureshi, "Adapt: Mitigating idling errors in qubits via adaptive dynamical decoupling," in *MICRO*, 2021.

[21] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi, "A case for multi-programming quantum computers," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 291–303.

[22] P. Das, S. S. Tannu, and M. Qureshi, "Jigsaw:boosting fidelity of nisq programs via measurement subsetting," in *MICRO*, 2021.

[23] S. Deshpande, C. Xu, T. Trochatos, Y. Ding, and J. Szefer, "Towards an antivirus for quantum computers," *arXiv preprint arXiv:2203.02649*, 2022.

[24] C. Developers, "Cirq," Apr. 2022, See full list of authors on Github: https://github .com/quantumlib/Cirq/graphs/contributors. [Online]. Available: https://doi.org/10.5281/zenodo.6599601

[25] Y. Ding, A. Holmes, A. Javadi-Abhari, D. Franklin, M. Martonosi, and F. Chong, "Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 828–840.

[26] J. Emerson, R. Alicki, and K. Życzkowski, "Scalable noise estimation with random unitary operators," *Journal of Optics B: Quantum and Semiclassical Optics*, vol. 7, no. 10, p. S347, 2005.

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

[28] X. Fu, M. Rol, C. Bultink, J. Van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, J. De Sterke, W. Vlothuizen, R. Schouten *et al.*, "An experimental microarchitecture for a superconducting quantum processor," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 813–825.

[29] P. Gokhale, Y. Ding, T. Propson, C. Winkler, N. Leung, Y. Shi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Partial compilation of variational algorithms for noisy intermediate-scale quantum machines," in *MICRO*. ACM, 2019, pp. 266–278.

[30] P. Gokhale, A. Javadi-Abhari, N. Earnest, Y. Shi, and F. T. Chong, "Optimized quantum compilation for near-term algorithms with openpulse," in *MICRO*. IEEE, 2020, pp. 186–200.

[31] A. Gold, J. Paquette, A. Stockklauser, M. J. Reagor, M. S. Alam, A. Bestwick, N. Didier, A. Nersisyan, F. Oruc, A. Razavi *et al.*, "Entanglement across separate silicon dies in a modular superconducting qubit device," *npj Quantum Information*, vol. 7, no. 1, pp. 1–10, 2021.

[32] Google, "Cirq," Accessed: November 19, 2019, https://github.com/quantumlib/Cirq.

[33] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: a scalable quantum programming language," in *ACM SIGPLAN Notices*, vol. 48, no. 6. ACM, 2013, pp. 333–342, https://www.mathstat.dal.ca/~selinger/quipper/.

[34] D. M. Greenberger, M. A. Horne, and A. Zeilinger, "Going beyond Bell's theorem," in *Bell's theorem, quantum theory and conceptions of the universe*. Springer, 1989, pp. 69–72.

[35] P. C. Haljan, K.-A. Brickman, L. Deslauriers, P. J. Lee, and C. Monroe, "Spin-dependent forces on trapped ions for phase-stable quantum gates and entangled states of spin and motion," *Phys. Rev. Lett.*, vol. 94, p. 153602, Apr 2005. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.94.153602

[36] J. Heckey, S. Patil, A. JavadiAbhari, A. Holmes, D. Kudrow, K. R. Brown, D. Franklin, F. T. Chong, and M. Martonosi, "Compiler management of communication and parallelism for quantum computation," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1. ACM, 2015, pp. 445–456.

[37] S. S. Hong, A. T. Papageorge, P. Sivarajah, G. Crossman, N. Didier, A. M. Polloreno, E. A. Sete, S. W. Turkowski, M. P. da Silva, and B. R. Johnson, "Demonstration of a parametrically activated entangling gate protected from flux noise," *Physical Review A*, vol. 101, no. 1, p. 012302, 2020.

[38] Y. Huang, S. Holtzen, T. Millstein, G. Van den Broeck, and M. Martonosi, "Logical abstractions for noisy variational quantum algorithm simulation," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 456–472.

[39] Y. Huang and M. Martonosi, "Qdb: from quantum algorithms towards correct quantum programs," *arXiv preprint arXiv:1811.05447*, 2018.

[40] Y. Huang and M. Martonosi, "Statistical assertions for validating patterns and finding bugs in quantum programs," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 541–553.

[41] IBM, "Quantum Software Development Kit for writing quantum computing experiments, programs, and applications," https://github.com/QISKit/qiskit-sdk-py#license, 2017, [Online; accessed 3-April-2018].

[42] IBM, "Measurement error mitigation," https://qiskit.org/textbook/ch-quantum-hardware/measurement-error-mitigation.html, 2020.

[43] IBM, "IBM Quantum," https://quantum-computing.ibm.com/, 2021.

[44] IBM, "IBM's roadmap for scaling quantum technology," 2021, https://research.ibm.com/blog/ibm-quantum-roadmap.

[45] IBM, "Novel coupling for rip gate based devices," https://research.ibm.com/publications/novel-coupling-for-rip-gate-based-devices, 2021.

[46] IBM, "Expanding the IBM Quantum roadmap to anticipate the future of quantum-centric supercomputing," 2022, https://research.ibm.com/blog/ibm-quantum-roadmap-2025.

[47] IBM, "IBM Unveils New Roadmap to Practical Quantum Computing Era; Plans to Deliver 4,000+ Qubit System ," 2022, https://newsroom.ibm.com/2022-05-10-IBM-Unveils-New-Roadmap-to-Practical-Quantum-Computing-Era-Plans-to-Deliver-4,000-Qubit-System.

[48] IBM, "With fault tolerance the ultimate goal, error mitigation is the path that gets quantum computing to usefulness," 2022, https://research.ibm.com/blog/gammabar-for-quantum-advantage.

[49] A. Javadi-Abhari, P. Gokhale, A. Holmes, D. Franklin, K. R. Brown, M. Martonosi, and F. T. Chong, "Optimized surface code communication in superconducting quantum computers," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 692–705.

[50] P. Jurcevic, A. Javadi-Abhari, L. S. Bishop, I. Lauer, D. F. Bogorin, M. Brink, L. Capelluto, O. Günlük, T. Itoko, N. Kanazawa *et al.*, "Demonstration of quantum volume 64 on a superconducting quantum computing system," *Quantum Science and Technology*, 2021.

[51] M. Khezri, J. Dressel, and A. N. Korotkov, "Qubit measurement error from coupling with a detuned neighbor in circuit qed," *Physical Review A*, vol. 92, no. 5, p. 052306, 2015.

[52] N. Killoran, J. Izaac, N. Quesada, V. Bergholm, M. Amy, and C. Weedbrook, "Strawberry fields: A software platform for photonic quantum computing," *Quantum*, vol. 3, p. 129, 2019, https://strawberryfields.readthedocs.io/en/latest/.

[53] P. V. Klimov, J. Kelly, J. M. Martinis, and H. Neven, "The snake optimizer for learning quantum processor control parameters," *arXiv preprint arXiv:2006.04594*, 2020.

[54] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland, "Randomized benchmarking of quantum gates," *Physical Review A*, vol. 77, no. 1, p. 012307, 2008.

[55] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, no. 2, p. 021318, 2019.

[56] L. Lao, P. Murali, M. Martonosi, and D. Browne, "Designing calibration and expressivity-efficient instruction sets for quantum computing," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 846–859.

[57] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation," *arXiv preprint arXiv:2005.13018*, 2020.

[58] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *ASPLOS*, 2019, pp. 1001–1014.

[59] G. Li, A. Wu, Y. Shi, A. Javadi-Abhari, Y. Ding, and Y. Xie, "Paulihedral: a generalized block-wise compiler optimization framework for quantum simulation kernels," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 554–569.

[60] J. Liu, L. Bello, and H. Zhou, "Relaxed peephole optimization: A novel compiler optimization for quantum circuits," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2021, pp. 301–314.

[61] J. Liu, G. T. Byrd, and H. Zhou, "Quantum circuits for dynamic runtime assertions in quantum computation," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1017–1030.

[62] J. Liu, P. Li, and H. Zhou, "Not all swaps have the same cost: A case for optimization-aware qubit routing," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 709–725.

[63] A. Lye, R. Wille, and R. Drechsler, "Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits," in *ASPDAC*. IEEE, 2015, pp. 178–183.

[64] L. S. Madsen, F. Laudenbach, M. F. Askarani, F. Rortais, T. Vincent, J. F. Bulmer, F. M. Miatto, L. Neuhaus, L. G. Helt, M. J. Collins *et al.*, "Quantum computational advantage with a programmable photonic processor," *Nature*, vol. 606, no. 7912, pp. 75–81, 2022.

[65] E. Magesan, J. Gambetta, and J. Emerson, "Robust randomized benchmarking of quantum processes," *arXiv preprint arXiv:1009.3639*.

[66] M. Martonosi and M. Roetteler, "Next steps in quantum computing: Computer science's role," *arXiv preprint arXiv:1903.10541*, 2019.

[67] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne, "Quantum circuit simplification and level compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 436–444, 2008.

[68] A. J. McCaskey, D. I. Lyakh, E. F. Dumitrescu, S. S. Powers, and T. S. Humble, "Xacc: A system-level software infrastructure for heterogeneous quantum-classical computing," *arXiv preprint arXiv:1911.02452*, 2019, https://github.com/ORNL-QCI/qcor.

[69] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, p. 023023, 2016.

[70] A. Mi, S. Deng, and J. Szefer, "Short paper: Device-and locality-specific fingerprinting of shared nisq quantum computers," *arXiv preprint arXiv:2202.12731*, 2022.

[71] K. Mølmer and A. Sørensen, "Multiparticle entanglement of hot trapped ions," *Phys. Rev. Lett.*, vol. 82, pp. 1835–1838, Mar 1999. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.82.1835

[72] 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 *ASPLOS*, 2019.

[73] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Architecting noisy intermediate-scale quantum computers: A real-system study," *IEEE Micro*, vol. 40, no. 3, 2020.

[74] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Full-stack, real-system quantum computer studies: architectural comparisons and design insights," in *ISCA*, 2019.

[75] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," in *ASPLOS*, 2020.

[76] G. Nannicini, L. S. Bishop, O. Gunluk, and P. Jurcevic, "Optimal qubit assignment and routing via integer programming," *arXiv preprint arXiv:2106.06446*, 2021.

[77] A. Oddi and R. Rasconi, "Greedy randomized search for scalable compilation of quantum circuits," in *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2018, pp. 446–461.

[78] N. A. of Sciences Engineering and Medicine, *Quantum Computing: Progress and Prospects*, E. Grumbling and M. Horowitz, Eds. Washington, DC: The National Academies Press, 2019.

[79] H. Paik, A. Mezzacapo, M. Sandberg, D. T. McClure, B. Abdo, A. D. Córcoles, O. Dial, D. F. Bogorin, B. L. T. Plourde, M. Steffen, A. W. Cross, J. M. Gambetta, and J. M. Chow, "Experimental demonstration of a resonator-induced phase gate in a multiqubit circuit-qed system," *Phys. Rev. Lett.*, vol. 117, p. 250502, Dec 2016. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.117.250502

[80] T. Patel, D. Silver, and D. Tiwari, "Geyser: a compilation framework for quantum computing with neutral atoms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 383–395.

[81] T. Patel and D. Tiwari, "Veritas: accurately estimating the correct output on noisy intermediate-scale quantum computers," in *SC*. IEEE, 2020.

[82] T. Patel and D. Tiwari, "Qraft: reverse your quantum circuit and know the correct program output," in *ASPLOS*, 2021.

[83] T. Patel, E. Younis, C. Iancu, W. de Jong, and D. Tiwari, "Robust and resource-efficient quantum circuit approximation," *arXiv preprint arXiv:2108.12714*, 2021.

[84] T. Patel, E. Younis, C. Iancu, W. de Jong, and D. Tiwari, "Quest: systematically approximating quantum circuits for higher output fidelity," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 514–528.

[85] T. Patel, E. Younis, C. Iancu, W. de Jong, and D. Tiwari, "Robust quantum circuit approximation for resource-efficient circuit synthesis," *Bulletin of the American Physical Society*, 2022.

[86] J. Paykin, R. Rand, and S. Zdancewic, "Qwire: a core language for quantum circuits," in *ACM SIGPLAN Notices*, vol. 52, no. 1. ACM, 2017, pp. 846–858, https://github.com/inQWIRE/QWIRE.

[87] E. Pelofske, A. Bärtschi, and S. Eidenbenz, "Quantum volume in practice: What users can expect from nisq devices," *arXiv preprint arXiv:2203.03816*, 2022.

[88] A. D. (Pennylane), "A brief overview of VQE," 2022, https://pennylane.ai/qml/demos/tutorial_vqe.html.

[89] E. C. Peterson, L. S. Bishop, and A. Javadi-Abhari, "Optimal synthesis into fixed xx interactions," *Quantum*, vol. 6, p. 696, 2022.

[90] J. Preskill, "Quantum computing in the nisq era and beyond," *arXiv preprint arXiv:1801.00862*, 2018.

[91] G. S. Ravi, K. N. Smith, P. Gokhale, and F. T. Chong, "Quantum computing in the cloud: Analyzing job and machine characteristics," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2021, pp. 39–50.

[92] G. S. Ravi, K. N. Smith, P. Gokhale, A. Mari, N. Earnest, A. Javadi-Abhari, and F. T. Chong, "Vaqem: A variational approach to quantum error mitigation," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 288–303.

[93] G. S. Ravi, K. N. Smith, P. Murali, and F. T. Chong, "Adaptive job and resource management for the growing quantum cloud," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2021, pp. 301–312.

[94] M. Reagor, C. B. Osborn, N. Tezak, A. Staley, G. Prawiroatmodjo, M. Scheer, N. Alidoust, E. A. Sete, N. Didier, M. P. da Silva *et al.*, "Demonstration of universal parametric entangling gates on a multi-qubit lattice," *Science advances*, vol. 4, no. 2, p. eaao3603, 2018.

[95] K. Reuer, J.-C. Besse, L. Wernli, P. Magnard, P. Kurpiers, G. J. Norris, A. Wallraff, and C. Eichler, "Realization of a universal quantum gate set for itinerant microwave photons," *Physical Review X*, vol. 12, no. 1, p. 011008, 2022.

[96] Rigetti, "The Quil Compiler," 2021, https://pyquil-docs.rigetti.com/en/stable/compiler.html.

[97] M. Sarovar, T. Proctor, K. Rudinger, K. Young, E. Nielsen, and R. Blume-Kohout, "Detecting crosstalk errors in quantum information processors," *Quantum*, vol. 4, p. 321, 2020.

[98] E. A. Sete, N. Didier, A. Q. Chen, S. Kulshreshtha, R. Manenti, and S. Poletto, "Parametric-resonance entangling gates with a tunable coupler," *Physical Review Applied*, vol. 16, no. 2, p. 024050, 2021.

[99] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum-logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1000–1010, 2006.

[100] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Optimized compilation of aggregated instructions for realistic quantum computers," in *ASPLOS*, 2019.

[101] D. Silver, T. Patel, and D. Tiwari, "Quilt: Effective multi-class classification on quantum computers using an ensemble of diverse quantum classifiers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8324–8332.

[102] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *CGO*. ACM, 2018, pp. 113–125.

[103] K. N. Smith, G. S. Ravi, P. Murali, J. M. Baker, N. Earnest, A. Javadi-Abhari, and F. T. Chong, "Error mitigation in quantum computers through instruction scheduling," *arXiv preprint:2105.01760*, 2021.

[104] R. S. Smith, M. J. Curtis, and W. J. Zeng, "A practical quantum instruction set architecture," *arXiv preprint arXiv:1608.03355*, 2016.

[105] D. S. Steiger, T. Häner, and M. Troyer, "Projectq: an open source software framework for quantum computing," *Quantum*, vol. 2, no. 49, pp. 10–22 331, 2018, https://projectq.ch/.

[106] S. Stein, N. Wiebe, Y. Ding, J. Ang, and A. Li, "Quantum bayesian error mitigation employing poisson modelling over the hamming spectrum for quantum error mitigation," *arXiv e-prints*, pp. arXiv–2207, 2022.

[107] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li, "Eqc: ensembled quantum computing for variational quantum algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 59–71.

[108] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, "Q#: Enabling scalable quantum computing and development with a high-level dsl," in *Proceedings of the Real World Domain Specific Languages Workshop 2018*. ACM, 2018, p. 7.

[109] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3400302.3415620

[110] W. Tang, T. Tomesh, M. Suchara, J. Larson, and M. Martonosi, "Cutqc: using small quantum computers for large quantum circuit evaluations," in *ASPLOS*, 2021.

[111] S. Tannu, P. Das, R. Ayanzadeh, and M. Qureshi, "Hammer: boosting fidelity of noisy quantum circuits by exploiting hamming behavior of erroneous outcomes," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 529–540.

[112] S. S. Tannu, Z. A. Myers, P. J. Nair, D. M. Carmean, and M. K. Qureshi, "Taming the instruction bandwidth of quantum computers via hardware-managed error correction," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 679–691.

[113] S. S. Tannu and M. K. Qureshi, "A case for variability-aware policies for nisq-era quantum computers," *arXiv preprint arXiv:1805.10224*, 2018.

[114] 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 *ASPLOS*, 2019.

[115] B. Villalonga, D. Lyakh, S. Boixo, H. Neven, T. S. Humble, R. Biswas, E. G. Rieffel, A. Ho, and S. Mandrà, "Establishing the quantum supremacy frontier with a 281 pflop/s simulation," *arXiv preprint arXiv:1905.00444*, 2019.

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

[117] H. Wang, J. Gu, Y. Ding, Z. Li, F. T. Chong, D. Z. Pan, and S. Han, "Quantumnat: Quantum noise-aware training with noise injection, quantization and normalization," *arXiv preprint arXiv:2110.11331*, 2021.

[118] D. Wecker and K. M. Svore, "Liqui—¿: A software design architecture and domain-specific language for quantum computing," *arXiv preprint arXiv:1402.4467*, 2014.

[119] M. Weiden, J. Kalloor, T. Patel, E. Younis, C. Iancu, and J. Kubiatowicz, "Topology aware unitary synthesis for scalable quantum circuit optimization," *Bulletin of the American Physical Society*, 2022.

[120] Wikipedia, "Total Variation Distance," https://en.wikipedia.org/wiki/Total_variation_distance_of_probability_measures, 2020.

[121] Wikipedia, "Mølmer–sørensen gate," https://en.wikipedia.org/wiki/M%C3%B8lmer%E2%80%93S%C3%B8rensen_gate, 2021.

[122] Wikipedia, "Operator norm," https://en.wikipedia.org/wiki/Operator_norm, 2022.

[123] Wikipedia, "Spearman's rank correlation coefficient," https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient, 2022.

[124] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in *ISMVL*. IEEE, 2008, pp. 220–225.

[125] R. Wille, A. Lye, and R. Drechsler, "Optimal swap gate insertion for nearest neighbor quantum circuits," in *ASPDAC*. IEEE, 2014.

[126] A. Wu, G. Li, Y. Wang, B. Feng, Y. Ding, and Y. Xie, "Towards efficient ansatz architecture for variational quantum algorithms," *arXiv preprint arXiv:2111.13730*, 2021.

[127] Y. Wu, W.-S. Bao, S. Cao, F. Chen, M.-C. Chen, X. Chen, T.-H. Chung, H. Deng, Y. Du, D. Fan *et al.*, "Strong quantum computational advantage using a superconducting quantum processor," *Physical Review Letters*, vol. 127, no. 18, p. 180501, 2021.

[128] M. Xu, Z. Li, O. Padon, S. Lin, J. Pointing, A. Hirth, H. Ma, J. Palsberg, A. Aiken, U. A. Acar *et al.*, "Quartz: superoptimization of quantum circuits," in *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2022, pp. 625–640.

[129] E. Younis, K. Sen, K. Yelick, and C. Iancu, "Qfast: Conflating search and numerical optimization for scalable quantum circuit synthesis," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2021, pp. 232–243.

[130] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *ASPLOS*, 2021, pp. 360–374.

[131] H. Zhou and G. T. Byrd, "Quantum circuits for dynamic runtime assertions in quantum computation," *IEEE Computer Architecture Letters*, vol. 18, no. 2, pp. 111–114, 2019.

[132] A. Zulehner, A. Paler, and R. Wille, "Efficient mapping of quantum circuits to the ibm qx architectures," in *DATE*. IEEE, 2018.