DisQ: A Novel Quantum Output State Classification Method on IBM Quantum Computers using OpenPulse

Tirthak Patel
Northeastern University

Devesh Tiwari
Northeastern University

ABSTRACT

Superconducting quantum computing technology has ushered in a new era of computational possibilities. While a considerable research effort has been geared toward improving the quantum technology and building the software stack to efficiently execute quantum algorithms with reduced error rate, effort toward optimizing how quantum output states are defined and classified for the purpose of reducing the error rate is still limited. To this end, this paper proposes DisQ, a quantum output state classification approach which reduces error rates of quantum programs on NISQ devices.

1 INTRODUCTION

Quantum computing is advancing at a rapid pace with the proliferation of different quantum computing technologies and renewed interest from industry and academia. In addition to the better established quantum annealing approach which has limited applicability [17, 30, 31], different quantum computing technologies are being actively explored to build a reliable quantum bit (or qubit), including superconducting qubits, trapped-ion qubits, and photon-based qubits. The most promising of these is the superconducting qubit technology which is primarily pioneered by IBM and Google. In fact, Google recently used their 53-qubit Sycamore quantum computing chip to run within a few seconds a task which would take a few days even on large supercomputers [3].

State-of-the-art Noisy Intermediate-Scale Quantum (NISQ) computers based on superconducting quantum technology are actively being used to establish the usefulness of quantum computers with potential applications ranging from chemistry and physics simulations to hard optimization problems [6, 18, 24, 33]. Unfortunately, due to the primitiveness of the technology, current NISQ computers have high error rates. Further, they do not have enough number of qubits to deploy meaningful error correction. Because of the high error rate of NISQ machines, the outputs generated by quantum algorithms run on current NISQ computers are erroneous.

A significant amount of research effort has been geared toward understanding, debugging, and mitigating the errors of these computers. These works have mainly focused on three primary research aspects: (1) developing quantum error-and-noise-aware simulation frameworks [2, 10, 26], (2) debugging quantum programs to identify code-level errors [23, 27], and (3) providing a best-effort solution to the NP-hard problem of mapping a logical quantum algorithm to a physical set of qubits such that the error rate of the produced output is minimized [4, 5, 14, 21, 22, 28, 34, 35, 37–39, 41, 43].

However, research efforts to address the problem of quantum output state classification are still in a nascent stage. This problem refers to the task of determining whether a qubit's output energy signal, after a quantum algorithm's execution, should be classified as the $|0\rangle$ state or the $|1\rangle$ state. This is not a straightforward task as the output state is affected by a variety of error-inducing factors.

To the best of our knowledge, we propose DisQ, the first method to optimize the classifier which differentiates between different quantum output states using the quantum pulse schedules on an IBM superconducting quantum computer. DisQ is built on two key insights: (1) The classification methodology, including choice of quantum gates and classifier shape, can affect the error rate of quantum output; (2) More surprisingly, the quantum output error rate is dependent on the probability of the output states. Current state-of-art classifier is not aware of these characteristics and hence, suffers from a relatively high error rate. DisQ addresses this challenge by building a classifier that is trained using multiple micro-benchmarks with different output state probabilities and optimized using a simulated annealing approach. We proposes two versions of DisQ: circle- and ellipse-shaped classifiers. Our evaluation shows that DisQ improves the median error rate, the 75th percentile error rate and the variability in error rate. DisQ is practical and suitable for deployment: the training data can be collected within 2 minutes during calibration (calibration is the task of determining the optimal parameters to drive the qubit) and the classifier can be optimized within 7 minutes on an Intel Core i7 processor, and can be used for all algorithm runs until the next calibration period (approx. every 12 hours).

DisQ's classification toolbox is compatible with IBM's Python-based Qiskit OpenPulse framework and can be used with any IBM quantum computer which supports OpenPulse. It is available at: http://github.com/GoodwillComputingLab/DISQ.

2 QUANTUM COMPUTING BACKGROUND

In this section, we provide some background of quantum computing, superconducting quantum computers, and their sources of errors.

2.1 Quantum Computing Fundamentals

A *qubit* is the building block of a quantum computer. A classical bit can exist in one of two states: 0 or 1. On the other hand, a *qubit's state* ($|\Psi\rangle$), during computation, can be expressed as a *superposition* of
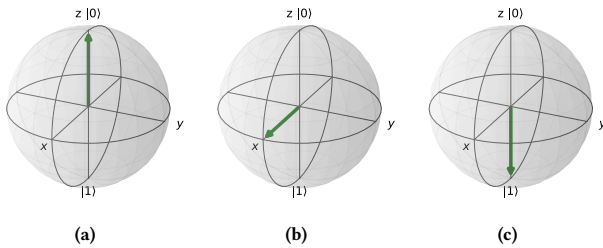
Figure 1: The Block Sphere visualizes a qubit's state. Here, two $R_y(\pi/2)$ gates are applied to a qubit initialized to state $|0\rangle$.
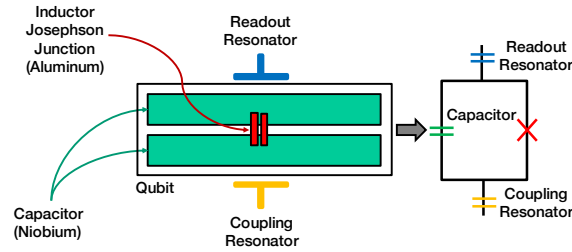

Figure 2: A superconducting qubit is made out of an LC oscillator circuit constructed using a capacitor and a Josephson Junction.

the two basis states: $|0\rangle$ and $|1\rangle$ (in bra-ket notation). More formally, $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers such that $\|\alpha\|^2 + \|\beta\|^2 = 1$. When the quantum computation is completed, the qubit's state is measured or *read out*. When the qubit is read out, its superposition is destroyed, and it is measured in state $|0\rangle$ (with probability $\|\alpha\|^2$) or in state $|1\rangle$ (with probability $\|\beta\|^2$).

A *quantum algorithm* is a set of *quantum gates* sequentially applied to the qubits on a quantum computer. When a quantum algorithm completes execution, the state of all or some of the qubits is measured to analyze the output. A qubit's state can be read out only once during the algorithm execution as it collapses the qubit's superposition. Therefore, it is only read out at the end of the algorithm's execution. Note that the output of a quantum algorithm is probabilistic. Therefore, multiple *trials* are conducted to get the *output probabilities* of a qubit state. For example, if a qubit has state $|\Psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}\beta|1\rangle$, then the output probability of state $|0\rangle$ is $\left\|\frac{1}{\sqrt{2}}\right\|^2 = \frac{1}{2}$ and state $|1\rangle$ is also $\left\|\frac{1}{\sqrt{2}}\right\|^2 = \frac{1}{2}$. Therefore, if 1024 trials are conducted, then it would be expected that half of the trials (512) would have state $|0\rangle$ and the other half would have state $|1\rangle$.

Quantum gates can be of 1-qubit or 2-qubit variety. A 1-qubit gate operates on a single qubit and puts it in the desired superposition. A 2-qubit gate *entangles* two qubits and modifies the superposition of the *target* qubit based on the superposition of the *control* qubit. A general 1-qubit gate has three components: x-, y-, and z- rotation components which rotate the qubit about the x-axis, y-axis and z-axis, respectively, on the Bloch Sphere (denoted as ($R_x(\theta)$, $R_y(\phi)$, and $R_z(\delta)$, respectively, where θ, ϕ, and δ are the angles of rotation). The Bloch Sphere is a unit sphere with the $|0\rangle$ state represented as a vector pointing toward the positive z-axis and the $|1\rangle$ state represented on the negative z-axis. While, the qubit state vector can be pointed in any direction on the Bloch Sphere, when it is measured, it collapses to the $|0\rangle$ or the $|1\rangle$ state, and its
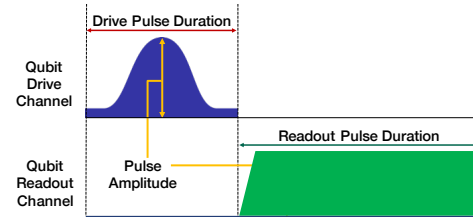

Figure 3: Microwave pulses are applied to drive the qubit to a desired state and read out the qubit value.

output probability is the projection of the vector onto the z-axis. For example, in Fig. 1, the Bloch Sphere shows the state changes after applying gates to a qubit initialized to state $|\psi\rangle = |0\rangle$. First, a $R_y(\pi/2)$ gate is applied. This puts the qubit in equal superposition (0.5 probability of observing both $|0\rangle$ and $|1\rangle$). Another $R_y(\pi/2)$ gate brings the qubit to state $|1\rangle$. Arbitrary rotations can be applied about any of the three axes to achieve the desired superposition. All 1-qubit rotations have 2-qubit variants ($CR_x(\theta)$, $CR_y(\phi)$ and $CR_z(\delta)$), where the target qubit is rotated as per the control qubit.

2.2 IBM's Superconducting Qubit Technology

IBM uses a circuit-based approach toward developing qubits using superconducting Josephson Junctions. As shown in Fig. 2, a superconducting niobium linear capacitor and a superconducting aluminium Josephson Junction, which behaves as a non-linear non-dissipative inductor, are developed on a silicon wafer. To build the Josephson Junction, two pieces of weakly coupled superconducting electrodes are separated by a very thin tunnel barrier which serves as an insulating layer. Overall, this forms a non-linear LC oscillator which behaves like a qubit if the parameters are tuned correctly. The oscillator allows for a two-level quantum system with discrete quantum energy levels. The lowest two energy levels are used as the $|0\rangle$ (ground level) and the $|1\rangle$ (first excited level) states.

2.3 Driving and Measuring Qubit States

The superconducting-qubit system can be addressed using external controls. Typically, the superconducting qubit has a frequency in the range of 4-5 GHz. As shown in Fig. 3, a Microwave tone can be applied at the qubit frequency to drive the qubit. By applying a pulse shape to the Microwave tone, quantum gates can be achieved. Typically, a Microwave pulse with a Gaussian shape is used to drive the qubit. Frame-of-reference change is used to apply gates along different axes. The pulse with the maximum amplitude magnitude is known as the "π Pulse" and it is used to apply the $R_x(\pi)$ rotation, which transforms the $|0\rangle$ state to the $|1\rangle$ state and vice versa.

The qubit state is measured by coupling the qubit to a superconducting transmission resonator. It is ensured that the readout resonator frequency is dependent on the state of the qubit. Thus, the qubit state can be determined by probing the resonant frequency. Fig. 3 shows that a long Square Gaussian pulse is applied to measure the state of the qubit. Lastly, as shown in Fig. 2, 2-qubit entangling gates are applied using a superconducting coupling bus resonator.

2.4 Sources of Error in Quantum Computers

There are several sources of error in NISQ technology. Once initialized, a qubit can only hold the coherence of its state for a limited
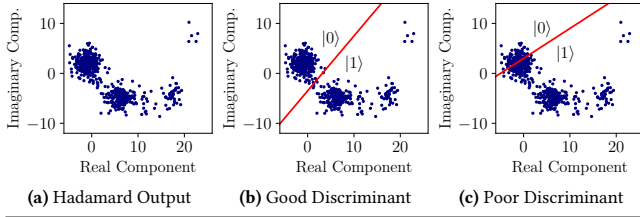
Figure 4: The role of a classifier is to identify the outputs of trials as $|0\rangle$ or $|1\rangle$ in a manner which minimizes the error.
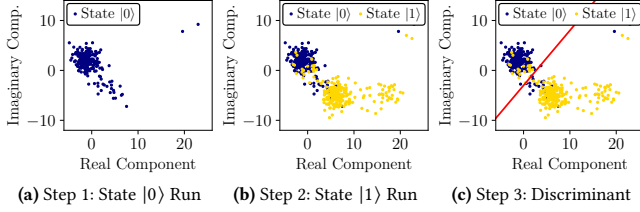

Figure 5: In the baseline case, a linear discriminant is used to differentiate the $|0\rangle$ vs. $|1\rangle$ using a three-step process.

amount of time. There are two types of coherence decays: (1) The T1 coherence refers to the amplitude damping. (2) The T2 coherence refers to the phase damping. NISQ errors also include the gate and readout errors. An erroneous application of the microwave pulses can cause gate errors, i.e., the gate could be put in a slightly-off-the-desired superposition which can lead to incorrect output probabilities. The readout resonators are also error-prone and cause readout errors. Refer to Patel et al. [32] for more details about the different sources of error. These errors can be a result of deviations from the optimal pulse parameters such as the frequency at which the pulse is applied, its amplitude, and its duration. Moreover, because the technology is still maturing, the qubit's properties, such as its frequency, vary. Therefore, the optimal pulse parameters need to be determined on a regular basis.

For this reason, IBM's computers are calibrated twice a day, and the qubit's error rates change after each calibration. *Calibration* is the task of determining qubit properties, such as frequency, and accordingly, setting the parameters of the microwave pulses for the gate operations. These parameters are then used to perform all the operations until the next calibration. Also, during the calibration period, the classifier which distinguishes between the $|0\rangle$ and the $|1\rangle$ state is developed (as discussed next in Sec. 3).

3 MOTIVATION FOR DISQ

In this section, we introduce the state classification problem, the existing state-of-art classification method, and its inefficiencies.

3.1 Classification of Output States

Once a quantum algorithm has completed execution, its final state is measured. To do this, the readout pulse is run on the readout channel (as shown in Fig. 3), and the corresponding signal, which measures the energy state of the qubit, is recorded on the acquire channel. The signal recorded during the entire acquire duration is then summed up and a single value is returned for each trial for each qubit. This value is a complex number with a real and an imaginary component. It is used to determine if the measured state represents the $|0\rangle$ state or the $|1\rangle$ state.

As an example, consider that a quantum gate known as the Hadamard gate is executed on a qubit initialized to state $|0\rangle$ and its resulting final state is read out. The resulting output values of conducting 1024 trials are shown in Fig. 4(a). The Hadamard gate puts a qubit in equal superposition of the $|0\rangle$ and the $|1\rangle$ state (0.5 probability of both states). Therefore, ideally, 512 of these trial outputs should be classified as state $|0\rangle$ and the other 512 as state $|1\rangle$. Given the nature of NISQ computers, the ideal classification is not possible. However, best effort must be made to reduce the *output error*. The output error (in percentage) is defined as the |*correct probability of state* $|0\rangle$ − *observed probability of state* $|0\rangle|\times100$ = |*correct probability of state* $|1\rangle$ − *observed probability of state* $|1\rangle|\times100$. The observed probability of $|0\rangle$ is calculated by dividing the number of trials which resulted in $|0\rangle$ by the total number of trials conducted. Also, by definition, probability of $|1\rangle = 1-$ probability of $|0\rangle$.

A trivial choice for state classification is a linear classifier. Consider the linear classifier used in Fig. 4(b) which classifies 548 trial outputs as state $|0\rangle$. Then, the observed output probability of state $|0\rangle$ is 548/1024 = 0.535, which results in an output error of $|0.5 - 0.535| \times 100 = 3.5\%$. A poorer quality classifier shown in Fig. 4(c) classifies 323 trial outputs as state $|0\rangle$, resulting in an error of 18.5%. Thus, developing a classifier which minimizes the output error is critical. Note that it is not possible to use different classifiers which cater to the output probabilities of different quantum algorithms as the output probabilities are not known for real quantum algorithms. Therefore, only one classifier is developed and used for all algorithms. Typically, the classifier is developed during calibration, once the qubit's frequency and pulse parameters, such as amplitude, are determined.

3.2 IBM's Existing Classification Methodology

Next, we look at the state-of-art method used for classification [2].

Step I. First, the qubit is initialized to the $|0\rangle$ state and its value is directly readout without running any gate operations. This is done for multiple trials. The probability of observing state $|0\rangle$ would be 1 in this case. Therefore, ideally, the output value of all trials should be classified as belonging to state $|0\rangle$. Fig. 5(a) shows the output spattering of 1024 trials of this run on the complex plane.

Step II. Next, the output values for the $|1\rangle$ state need to be generated. To achieve this, the qubit is initialized to the $|0\rangle$ state. Then, a π pulse (i.e., a rotation of π about the x-axis − $R_x(\pi)$) is applied to it to put it in the $|1\rangle$ state (pointing toward the negative z-axis in the Bloch Sphere). It is then measured, as shown in Fig. 3. Ideally, the output value of all trials of this run should be classified as belonging to the $|1\rangle$ state. Fig. 5(b) shows the output spattering of 1024 trials of this run which produces the $|1\rangle$ state.

Step III. The last step is to construct a discriminator. Fig. 5(b) shows that there could be some overlap between the values generated by the $|0\rangle$ state and the values generated by the $|1\rangle$ state. This is due to the various gate and coherence errors mentioned in Sec. 2 which contribute to erroneous output. Therefore, a perfect distinction between the $|0\rangle$ and the $|1\rangle$ state, which would result in an output error of 0%, is not possible to achieve. However, best effort must be made to reduce the output error. Currently, a linear
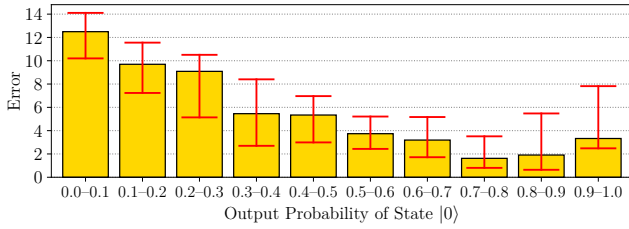
Figure 6: Different error rates are observed for runs with different output probabilities of state $|0\rangle$. The bars show the median in each bin; the range indicators show the spread from the 25th percentile error to the 75th percentile error.
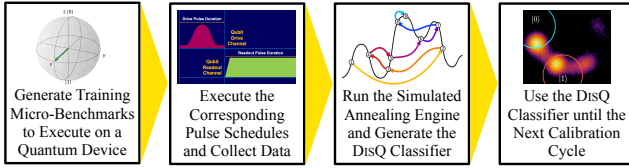

Figure 7: Overview of the steps when classifying states using DISQ.

discriminator is used to minimize this output error by generating a discriminator which maximizes the distance between the means of the classified samples, as shown in Fig. 5(c). Points that fall above the line are classified as $|0\rangle$ and ones that fall below the line are classified as $|1\rangle$. For all the runs conducted until the next calibration, this linear discriminator is used to classify the qubit state.

3.3 Inefficiencies of the Existing Method

The existing state classification method or the *baseline* method uses only two runs – one with output state $|0\rangle$ with probability 1 and the other with output state $|1\rangle$ with probability 1 – to develop the linear discriminant. As such, applying this discriminant to runs with other output probabilities can lead to several undesired consequences. To demonstrate this, we ran over one thousand micro-benchmark runs for a period of 10 days on IBM's 1-qubit Armonk quantum computer. For each run, a $U3(R_x(\theta), R_y(\phi), R_z(\delta))$ gate with three random rotation angles about each of the three axes, selected uniformly between $-\pi$ and π, is applied to a qubit initialized to state $|0\rangle$. This results in an arbitrary output probability of state $|0\rangle$ between 0 and 1. Fig. 6 shows the error rates of runs when they are binned in increments of 0.1 of their correct output probabilities. We ensured that all bins have the same number of runs. Fig. 6 shows several interesting results.

(1) The output error is highly correlated with the output probability of state $|0\rangle$. For example, the median error when the output probability of state $|0\rangle$ is 0.9-1 is 3.5% but the median error when the output probability of state $|0\rangle$ is 0-0.1 is 12.5%, which is 3.5× worse compared to the former's error rate. Some of this difference is due to the fact that when the output probability of state $|0\rangle$ is 0-0.1, it means that state $|1\rangle$ is expected for most of the trials. But, due to the coherence properties of qubits, state $|1\rangle$ is more likely to lose its coherence and drop from the first excited state to the ground state. However, a degradation in error rate of 3.5× is unacceptable. Quantum algorithms with lower output probability of state $|0\rangle$ should not observe a much higher error rate simply
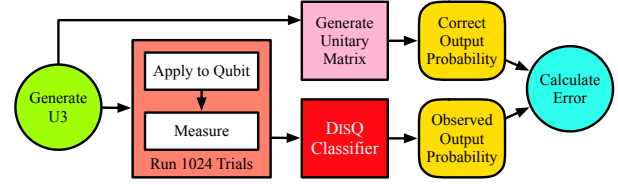

Figure 8: Processing and calculating error of a micro-benchmark.

due to the makeup of their output probabilities. An algorithm's output probability of states cannot be controlled or modified as it is an intrinsic property of the algorithm. Therefore, the output state classifier should be developed in accordance with these unintentional outcomes, such that similar error rates are observed regardless of the output probability of state $|0\rangle$.

(2) The error rate is high even for a single U3 gate used for the micro-benchmarks; a typical quantum algorithm consists of many gate operations and these errors get compounded. The median error across all the runs is about 5%, while the 75th percentile error is 9%. While the error rate is high in isolation, note that the error rate compounds when more gates are applied as each gate operation introduces its own error rate while also increasing the probability of state losing coherence as more time is spent before measuring the qubit state. Therefore, effort is needed to decrease the median error rate and especially, the 75th percentile error rate as 25% of the runs have more than 9% error rate.

(3) The spread or variability in error rate is high even among runs with similar output probabilities of state $|0\rangle$. For example, in Fig. 6, consider the 0.2-0.3 bin. It has a median error of 8.75%. But it has a significant error spread, i.e., the 75th percentile error - the 25th percentile error, of 6.5%. Similarly, other bins also have a high spread. The fact that the error rate varies so much even among runs with similar output probability of state $|0\rangle$ indicates that the error rates are not stable and the results are non-reproducible. Therefore, it needs to be ensured the spread of the error rates is minimized.

4 DISQ STATE CLASSIFICATION DESIGN

In this section, we describe DISQ, an approach to mitigate the aforementioned issues in Sec. 3. Fig. 7 provides a high-level overview of the procedure that DISQ uses to develop a discriminating state classifier. When a qubit is being calibrated, DISQ should be executed to obtain the state classifier. First, DISQ generates different micro-benchmarks with a diverse range of output probabilities of state $|0\rangle$ to cover the entire spectrum of possible output state probabilities. Next, it executes these micro-benchmarks on the qubit which needs to be calibrated and obtains the raw complex data. Post this, DISQ runs a black-box simulated annealing engine which optimizes the parameters of the discriminating classifier in a manner which minimizes the median error *and* the spread of the error of the training data. This classifier can then be used for all quantum algorithms executed on that qubit until the next calibration is performed.

4.1 Micro-Benchmark Design and Execution

The first order of business is to construct the micro-benchmarks whose output is fed to the state-classifier for discriminating the $|0\rangle$
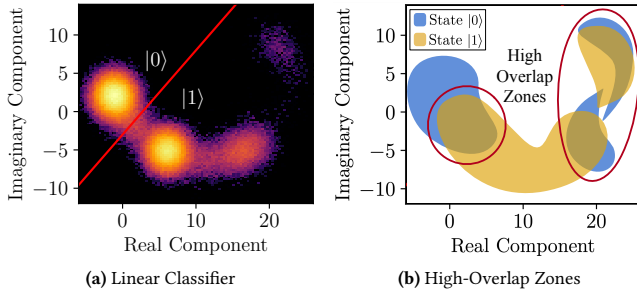
(a) Linear Classifier   **(b)** High-Overlap Zones

Figure 9: A linear classifier does not take into account high-overlap zones which are error-prone because they have a similar density of points belonging to the $|0\rangle$ and $|1\rangle$ states.
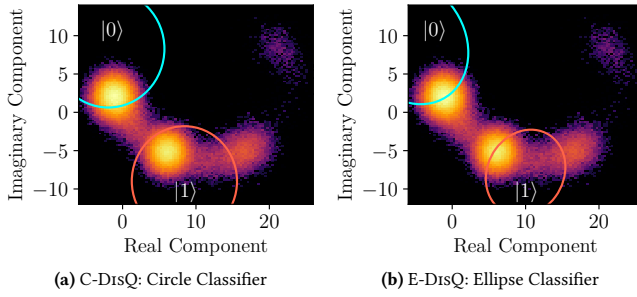

(a) C-DisQ: Circle Classifier   **(b)** E-DisQ: Ellipse Classifier

Figure 10: Visual representation of C-DisQ (circle classifier) and E-DisQ (ellipse classifier), which focus on low-overlap zones.

and the $|1\rangle$ states. The purpose of these micro-benchmarks is to cover the full spectrum of output probabilities of state $|0\rangle$ from 0 to 1. As discussed earlier, the reason for this purpose is to ensure that the developed classifier minimizes and achieves equal error rate for quantum algorithms with all types of output probabilities.

We achieve this on IBM's Qiskit framework by applying the $U3(R_x(\theta), R_y(\phi), R_z(\delta))$ gate with three random rotation angles about each of the three axes, selected uniformly between $-\pi$ and π, to a qubit initialized to state $|0\rangle$. This results in an arbitrary correct output probability of state $|0\rangle$ between 0 and 1 which can be calculated from the corresponding unitary matrix of the $U3$ gate. We run multiple such randomized micro-benchmarks. We ensure that each output probability of state $|0\rangle$ bin in increments of 0.1 (i.e., $0 - 0.1, 0.1 - 0.2, \ldots, 0.9 - 1$) contains equal number of micro-benchmarks. These micro-benchmarks are then executed on a quantum computer with 1024 trials each, their observed output probability is calculated using DisQ classifier, and their output error is calculated, as shown in Fig. 8. The next step for DisQ is to classify.

4.2 DisQ's Discriminating Classifiers

The primary consideration when developing a discriminating classifier is to determine the shape of the classifier. The existing approach is to use a linear discriminant as shown in Fig. 9(a), which shows an example density plot of the output of multiple training micro-benchmarks. The points above the line are classified as state $|0\rangle$, and the ones below are classified as state $|1\rangle$. Thus, a linear discriminant classifies points generated by all the trials that are conducted.

However, during the design of DisQ, our experimental results revealed that it might not be suitable to use all the trial outputs. As

Algorithm 1 DisQ's simulated annealing engine.

1: **Input:** Training data T_{data}, number of iteration N_{iter}, initial temperature T, and cooling coefficient α
2: Best configuration C_{best} (sampled randomly)
3: Best objective $P_{\text{best}} \Leftarrow \text{Objective}(T_{\text{data}}, C_{\text{best}})$
4: **for** $i = 1, \ldots, N_{\text{iters}}$ **do**
5:     Random neighbor configuration C_{new}.
6:     Corresponding $P_{\text{new}} \Leftarrow \text{Objective}(T_{\text{data}}, C_{\text{new}})$.
7:     Energy $E \leftarrow P_{\text{best}} - P_{\text{new}}$.
8:     **if** $E > 0$ **or** $E < 0$ **and** random() $< e^{E/T}$ **then**
9:         $C_{\text{best}} \Leftarrow C_{\text{new}}$
10:         $P_{\text{best}} \Leftarrow P_{\text{new}}$
11:     **end if**
12:     $T \Leftarrow \alpha \times T$
13: **end for**
14: **Output** Best discriminator configuration C_{best}

Fig. 9(b) shows, there are certain zones on the complex plane where both, points belonging to state $|0\rangle$ and points belonging to state $|1\rangle$, exist with an equal density. For example, 45% of the points falling in these regions could represent state $|0\rangle$ and the remaining 65% could represent state $|1\rangle$. Thus, these zones are not ideal for classification purposes as points in these regions are highly likely to be $|0\rangle$ or $|1\rangle$. *Informed by these observations, we designed DisQ to ignore such high-overlap zones and instead, focus on low-overlap zones.* For example, such low-overlap zones could be where point density of state $|0\rangle$ is 99% and that of state $|1\rangle$ is 1%.

A linear classifier cannot achieve this. In fact, any line-based classifier (quadratic, cubic, etc.) accounts for all trials by design. *To solve this problem, we propose two classifiers which can identify low-overlap zones and not consider high-overlap zones: a circle discriminant classifier (C-DisQ) and an ellipse discriminant classifier (E-DisQ).* Examples of both of these classifiers are given in Fig. 10. As shown in the figure, both of these classifiers can focus on low-overlap zones and avoid high-overlap zones. For example, consider an algorithm that is run with 1024 trials, and 400 fall in the $|0\rangle$ region, 500 fall in the $|1\rangle$ region, and 124 fall outside both regions. Then, those 124 trials can be ignored, and instead the observed output probability of state $|0\rangle$ can be calculated as $400/(400 + 500) = 0.44$. This is in contrast to a linear classifier where all trials have to be considered, even ones which fall in high-overlap error-prone zones.

However, it is non-trivial to determine the parameters which define the C-DisQ and E-DisQ classifiers. C-DisQ has six parameters, three for each circle (x-coordinate, y-coordinate, and radius). E-DisQ has ten parameters, five for each of the ellipses (x-coordinate, y-coordinate, width, height, and angle of rotation). While, C-DisQ has fewer parameters to optimize, E-DisQ has better precision over the classification region which can potentially lead to a lower error rate (both are evaluated in Sec. 5). Nonetheless, the problem of determining their parameters, while minimizing the error rate and error spread calculated based on the micro-benchmark measured data, is NP-hard. Therefore, next, we discuss the black-box simulated annealing engine used to optimize the two classifiers.

4.3 DisQ Simulated Annealing Engine

The goal of DisQ's simulated annealing engine is to obtain the *parameter configuration* for C-DisQ and E-DisQ such that the given objective is nearly-minimized (for the micro-benchmark measured

data). A parameter configuration is one permutation of six parameters for C-DisQ and ten parameters for E-DisQ, which define the characteristics of the circles or ellipses, respectively.

The first step is to design the optimization objective. One option is to minimize the median error of the training micro-benchmarks. However, just minimizing the median might create a scenario as shown in Fig. 6, where runs with a high output probability of state $|0\rangle$ have a much lower error rate than runs with a low output probability. Therefore, to avoid this scenario the spread of the error among the runs (i.e., spread is defined as the 75th percentile error − the 25th percentile error) is also minimized. However, if just the spread is chosen as the objective, then the median error might not be optimized (for example, spread can be minimized by choosing parameters such that all runs have an equally high error). *Therefore, to create a balance between the median and the spread, the objective that DisQ if to minimize the median error + the spread of the error* (we evaluate all three options in Sec. 5).

Next, Algorithm 1 shows how DisQ's simulated annealing engine iteratively steers toward a parameter configuration which minimizes the objective. Initially, the annealing environment has a high temperature, which means that it has a high likelihood of exploring different configuration neighborhoods. As the algorithm progresses, the temperature reduces and the algorithm narrows in on a near-optimal configuration neighborhood. At each iteration, a random configuration in the neighborhood of the current best configuration is sampled. A neighborhood is defined as all points within one unit for all of the parameters (six for C-DisQ and ten for E-DisQ). The objective of this sampled configuration is calculated for the training dataset. Depending on the current energy and temperature (as defined in Algorithm 1), it is determined if the best configuration should be updated to the newly sampled configuration. The best parameter configuration is obtained at the end of the execution, which can be used to classify the output of all succeeding quantum algorithm executions until the next calibration.

Scaling to multiple qubits. All qubits on a quantum computer need to be calibrated and separate classifiers need to be developed for all of them. The classification time overhead of DisQ does not increase with the number of qubits as every step of DisQ can run in parallel for all the qubits. This includes generating and executing the micro-benchmarks and independently running the simulated annealing engine concurrently.

5 DISQ EVALUATION

5.1 Evaluation Methodology

We use IBM's Armonk quantum computing machine (specifications are provided in Table 1) to conduct our experiments. While IBM has other machines available, Armonk is the only one which allows obtaining the pre-classification raw complex values (OpenPulse).

For demonstrating robustness, DisQ is evaluated and compared against IBM's existing state-of-art classifier over multiple days. The classifier's performance is evaluated using a validation dataset. The training and validation dataset are collected using the same methodology: 100 randomized U3-based micro-benchmarks, each with a different output probability of state $|0\rangle$, 10 belonging to each probability bin in increments of 0.1. In particular, DisQ is evaluated across ten days, with a classifier developed for each day (each

Table 1: Specifications of Armonk Quantum Computer.

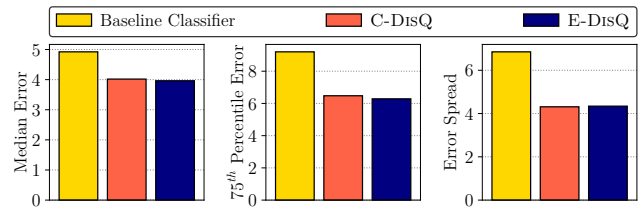| Online Date	10-16-2019
Number of Qubits	1
Drive Frequency	≈4.97 GHz
π Pulse Shape	Gaussian
π Pulse Amplitude	≈0.252 + 0.0j
π Pulse Duration	≈0.60 μs
Readout Frequency	≈6.99 GHz
Readout Pulse Shape	Square Gaussian
Readout Pulse Amplitude	≈0.605 + 0.0j
Readout Pulse Duration	≈3.52 μs

Figure 11: Both variants of DisQ achieve a lower median error, 75th percentile error, and error spread than the baseline classifier.

calibration). The two variants of DisQ, C-DisQ (circle classifier) and E-DisQ (ellipse classifier), are compared against the default Qiskit classifier, referred to as the baseline classifier. The metrics used include the median error (of the 100 validation micro-benchmarks across the 10 days), the 75th percentile error, and the error spread (the 75th percentile error − the 25th percentile error).

5.2 Evaluation Analysis

DisQ achieves a lower median error, 75th percentile error, and error spread than the baseline classifier. Fig. 11 shows that both C-DisQ and E-DisQ reduce the median error of the single-gate runs from 5% to 4%. Such a reduction can significantly improve the performance of quantum algorithms and programs with multiple gates. For instance, if five 5%-error gates are applied, the overall error rate would be $1 − (1 − 0.05)^5 = 23\%$. In comparison, if five 4%-error gates are applied, the overall error rate would be $1 − (1 − 0.04)^5 = 18\%$. Furthermore, the 75th percentile error is reduced from 9.5% to 6.5%. With the baseline classifier, 25% of runs have an error rate of more than 9.5%, while with DisQ, 75% of runs have an error rate of less than 6.5% and 50% of runs have an error of less than 4%. Note that E-DisQ performs slightly better than C-DisQ for the median error as well as the 75th percentile error because it enables a higher level of precision over the classification region.

Next, the error spread drops from 7% to ≈4.5% with C-DisQ and E-DisQ – a reduction of 36%. This indicates that not only does DisQ reduce the median error, but it also reduces the difference in the error rates observed by different runs, allowing for stable results. To follow up on this, next, we study how DisQ reduces the difference in the error rates of runs with different output probabilities.

DisQ achieves a better equalization of the error rate compared to the baseline classifier. Fig. 12 shows the median errors for runs with different output probability of state $|0\rangle$ for the baseline classifier, C-DisQ, and E-DisQ. Both C-DisQ and E-DisQ have similar performance, achieve a more equal distribution of error for all output probabilities, especially reducing the error rate of runs
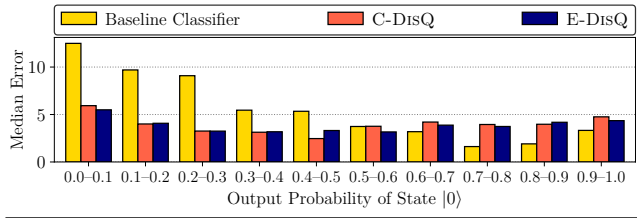
Figure 12: C-DisQ and E-DisQ achieve a more equal error rate across runs with output probabilities of state |0⟩ from 0 to 1.

(a) C-DisQ: Circle Classifier

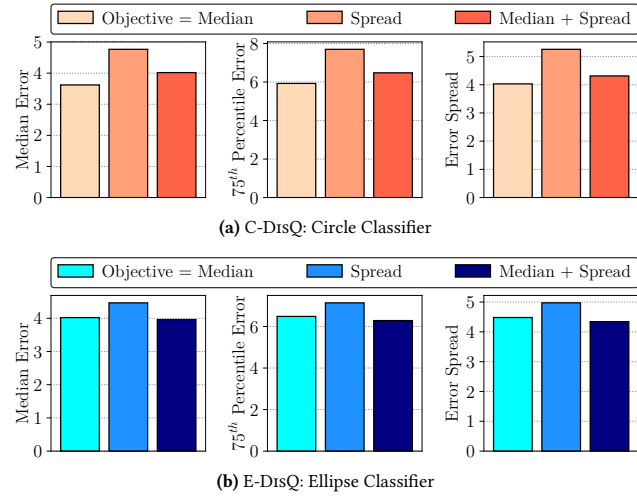**(b)** E-DisQ: Ellipse Classifier

Figure 13: Optimizing just the median achieves similar median error and error spread as optimizing the median + the spread. Setting the objective to just the error spread performs worse.

with output probability of state |0⟩ less than 0.5. For instance, the median error rate of the 0-0.1 bin is reduced from more than 12% to less than 6%. This is a reduction of more than 50% in the error rate which has a significant impact on the stability of output of the runs with output probability of state |0⟩ of less than 0.1. On the other hand, as a result of the distribution equalization, the error rates of runs with output probability of state |0⟩ of more than 0.6 have increased. However, this increase is comparably smaller than the decrease in error rate for other runs. Overall, DisQ achieves the goal of reducing the median error, while also equalizing the error rates across different outputs by also optimizing the error spread. Next, we look at the impact of optimizing different objective functions, including ones where the spread is not optimized.

Optimizing the median achieves similar low median error and error spread as optimizing the median + the spread. However, when the objective function is set to the median, day-to-day variability is much worse. Fig. 13(a) and (b) show the median error, the 75th percentile error, and the error spread for choosing different objectives for the simulated annealing engine to optimize C-DisQ and E-DisQ, respectively. Evidently, in both cases, optimizing just the spread gives worse performance than optimizing the other two metrics. In fact, even the error spread is higher by over 1% for C-DisQ and 0.5% for E-DisQ when the error spread is optimized. The reason for this is that because optimizing
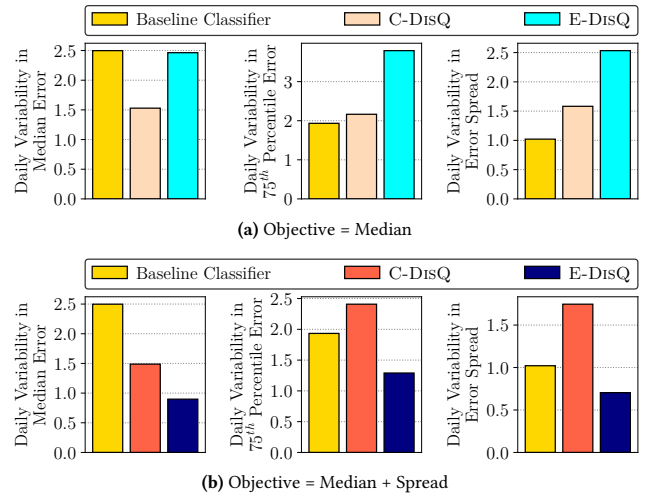

(a) Objective = Median

(b) Objective = Median + Spread

Figure 14: The day-to-day variability is lower with DisQ when the objective is to minimize the median + the spread. When just the median is minimized, day-to-day variability is much higher.

the error spread does not focus on optimizing the median error, the configuration neighborhoods where the median is optimized are not explored. But those neighborhoods have the potential to reduce the spread just by reducing the median, as we observe in the case when the median is optimized. Optimizing the median gives similar results as optimizing the median + the spread, with the latter performing slightly worse for C-DisQ and better for E-DisQ.

However, this does not indicate that it is advisable to optimize just the median. Fig. 14(a) and (b) show the daily variability in error metrics when just the median is optimized and when the median + the spread is optimized, respectively. The daily variability is calculated as the spread of a given metric across the 10 days. For example, if the median error is considered, the median error is calculated for the validation dataset for each of the 10 days, and then, the spread of those 10 samples is indicated in the figure. The baseline case has a daily variability of 2% error even in the case of the median error, which is a high variability. Interestingly, using the E-DisQ classifier with the median + the spread as the objective reduces that variability to less than 1%, which is a 50% reduction in error variability. This makes the error rates more stable from one day to another, just improving the reproducibility of the results. However, using the E-DisQ classifier with just the median as the objective has similar daily variability of median error as the baseline case. In fact, it has higher daily variability for the 75th percentile error and the error spread. This shows that while optimizing just the median error has good overall results, its daily variability is very high, making the results less reliable. On the other hand, DisQ with the median + the spread as the objective, can reduce the daily variability and improve the stability considerably.

Developing a new classifier every calibration achieves lower error rate and spread than using a fixed classifier across all calibrations, for both C-DisQ and E-DisQ. One might question the significance of developing a new classifier every calibration, after all, the output of the same qubit is being classified. However,
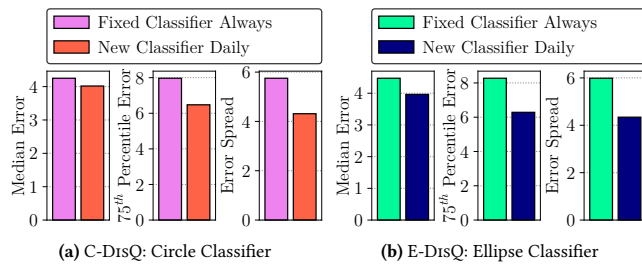
(a) C-DɪꜱQ: Circle Classifier **(b)** E-DɪꜱQ: Ellipse Classifier

Figure 15: Developing a new classifier every calibration achieves lower error rate than using the same classifier across all calibrations.
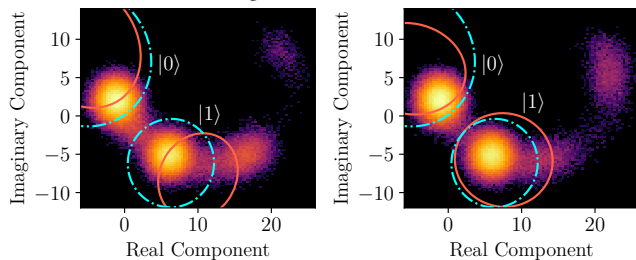

Figure 16: Difference in classifiers when a fixed classifier is used always (broken blue lines) vs. when a new classifier is developed every calibration (red solid lines).
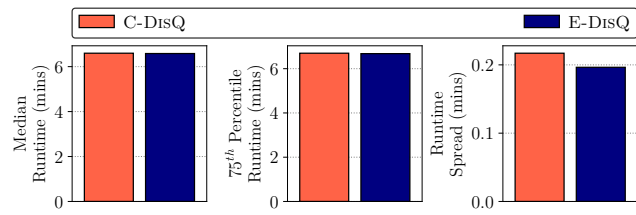

Figure 17: Both C-DɪꜱQ and E-DɪꜱQ classifiers have similar low runtimes which are very stable across different executions.

Fig. 15 shows the importance of developing a new classifier after each calibration. For both C-DɪꜱQ and E-DɪꜱQ, using a fixed classifier (the one which performs the best among all the ones generated over the 10 days) for all 10 calibrations performs worse than generating a new classifier after every calibration. For example, for C-DɪꜱQ using a fixed classifier has an error spread of 6%, while using a new classifier daily has an error spread of 4.3%.

The reason for this is that due to the instability of the qubits, the measurement points generated daily might have different densities, requiring different classifiers as shown in Fig. 16. Here, the broken blue line shows the fixed best classifier across the 10 days, while the red lines indicate the classifiers which were the best for the two days shown. Evidently, the regions considered by the fixed classifier are quite different than the regions considered by the day-specific classifiers, even though they have some intersecting portions. However, the disjoint portions are the ones which actually contribute to the higher error rate of the fixed classifier.

Lastly, both C-DɪꜱQ and E-DɪꜱQ classifiers have similar low runtimes for executing the simulated annealing engine. The overhead of running the micro-benchmarks on the quantum computer is very small as it can execute the entire batch of 100 runs,

each with 1024 trials (100k trials in total), within 2 minutes, which is comparable to the baseline case as most of the time is consumed in setting up and initializing the qubits. Once, the training dataset is generated, then the simulated annealing engine needs to be executed. Fig. 17 shows that the execution time of DɪꜱQ's simulated annealing engine is less than 7 minutes on our local 4.20 GHz Intel Core i7-7700K machine for both C-DɪꜱQ and E-DɪꜱQ and the spread of these runtimes is ≈0.2 minutes across all the executions. This overhead is not on the critical path as the quantum computer can resume regular computation after the 2 minutes to run the micro-benchmarks. Note that we found that executing more than 100 micro-benchmarks (e.g., 1000) does not reduce the error rates, but does incur a higher overhead of performing the optimization.

Even though E-DɪꜱQ optimizes 10 parameters, it has an equivalent runtime to C-DɪꜱQ, which optimizes 6 parameters. Thus, overall, E-DɪꜱQ not only has lower error rates, but also has comparable runtime to C-DɪꜱQ, demonstrating that it is a more suitable option.

6 RELATED WORK

Previous works have proposed qubit Quantum Error Correction (QEC) codes, which have an overhead of more than 10 physical qubits per logical qubit [7, 20, 36, 40]. These methods are thus untenable for current NISQ devices, which require a low-overhead (in terms of the number of ancillary qubits required) solution to reduce the error so that they can execute quantum algorithms.

On the other hand, a large amount of focus has been dedicated toward optimizing the execution of quantum algorithms. This includes developing frameworks and compilers to optimize the mapping of a quantum algorithm to a quantum computer such as IBM's Qiskit compiler and Google's Cirq framework [1, 2, 11, 16, 19]. In conjunction with these industry-led efforts, academic research has also focused on reducing the error rates of quantum algorithms by proposing debugging methods, simulation strategies, and noise-and-error-aware algorithm mapping approaches [4, 5, 8, 12, 14, 18, 21, 22, 25, 27–29, 32, 34, 35, 35, 38, 39, 41–43]. For example, Murali et al. [28] propose a method to reduce error rate by performing cross-talk-aware algorithm mapping to the qubits.

While these works focus on the higher-level problems of an algorithm's execution stack, IBM's OpenPulse, which is the framework to apply the pulses for quantum gates, has also been previously leveraged to solve problems including optimizing compilation [2, 9, 13, 15]. However, none of the above works have optimized the state classification problem by using raw output data of applying pulses, and thus, these works can be used in a compatible manner with DɪꜱQ to minimize the error rates of NISQ devices.

7 CONCLUSION

This paper presented DɪꜱQ, the first work to optimize state classifiers which differentiate between different quantum states. DɪꜱQ demonstrates that the output error and its variability on NISQ devices can be reduced just by optimizing the classification methodology, without any hardware or compiler modifications. DɪꜱQ is available at: `http://github.com/GoodwillComputingLab/DISQ`.

REFERENCES

[1] Talha Ahmed. 2018. *Q#: A Quantum Programming Language by Microsoft.* Ph.D. Dissertation. Imperial College London.

[2] G Aleksandrowicz, T Alexander, P Barkoutsos, L Bello, Y Ben-Haim, D Bucher, et al. [n.d.]. Qiskit: An Open-source Framework for Quantum Computing.(2019).

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum Supremacy using a Programmable Superconducting Processor. *Nature* 574, 7779 (2019), 505–510.

[4] Abdullah Ash-Saki, Mahabubul Alam, and Swaroop Ghosh. 2019. QURE: Qubit Re-allocation in Noisy Intermediate-Scale Quantum Computers. In *Proceedings of the 56th Annual Design Automation Conference 2019.* ACM, 141.

[5] Debjyoti Bhattacharjee, Abdullah Ash Saki, Mahabubul Alam, Anupam Chattopadhyay, and Swaroop Ghosh. 2019. MUQUT: Multi-Constraint Quantum Circuit Mapping on NISQ Computers. In *38th IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2019.* Institute of Electrical and Electronics Engineers Inc., 8942132.

[6] Sergey Bravyi, Graeme Smith, and John A Smolin. 2016. Trading Classical and Quantum Computational Resources. *Physical Review X* 6, 2 (2016), 021043.

[7] Jonathan J Burnett, Andreas Bengtsson, Marco Scigliuzzo, David Niepce, Marina Kudra, Per Delsing, and Jonas Bylander. 2019. Decoherence Benchmarking of Superconducting Qubits. *npj Quantum Information* 5, 1 (2019).

[8] Anastasiia Butko, George Michelogiannakis, Samuel Williams, Costin Iancu, David Donofrio, John Shalf, Jonathan Carter, and Irfan Siddiqi. 2019. Understanding Quantum Control Processor Capabilities and Limitations through Circuit Characterization. *arXiv preprint arXiv:1909.11719* (2019).

[9] Lauren Capelluto and Thomas Alexander. 2020. OpenPulse: Software for Experimental Physicists in Quantum Computing. *Bulletin of the American Physical Society* (2020).

[10] J Ignacio Cirac and Peter Zoller. 2012. Goals and Opportunities in Quantum Simulation. *Nature Physics* 8, 4 (2012), 264.

[11] Rigetti Computing. 2019. Pyquil documentation. *URL http://pyquil. readthedocs. io/en/latest* (2019).

[12] Poulami Das, Swamit S Tannu, Prashant J Nair, and Moinuddin Qureshi. 2019. A Case for Multi-Programming Quantum Computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 291–303.

[13] Eugen Dumitrescu, Raphael Pooser, and John Garmon. 2020. Benchmarking Noise Extrapolation with OpenPulse. *Bulletin of the American Physical Society* (2020).

[14] Pranav Gokhale, Yongshan Ding, Thomas Propson, Christopher Winkler, Nelson Leung, Yunong Shi, David I Schuster, Henry Hoffmann, and Frederic T Chong. 2019. Partial Compilation of Variational Algorithms for Noisy Intermediate-Scale Quantum Machines. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 266–278.

[15] Pranav Gokhale, Ali Javadi-Abhari, Nathan Earnest, Yunong Shi, and Frederic T Chong. 2020. Optimized Quantum Compilation for Near-Term Algorithms with OpenPulse. *arXiv preprint arXiv:2004.11205* (2020).

[16] Andrew Hancock, Austin Garcia, Jacob Shedenhelm, Jordan Cowen, and Calista Carey. [n.d.]. Cirq: A Python Framework for Creating, Editing, and Invoking Quantum Circuits. ([n. d.]).

[17] Mohamed W Hassan, Scott Pakin, and Wu-chun Feng. 2019. C to D-Wave: A High-level C Compilation Framework for Quantum Annealers. In *2019 IEEE High Performance Extreme Computing Conference (HPEC).* IEEE, 1–8.

[18] Yipeng Huang and Margaret Martonosi. 2019. Statistical Assertions for Validating Patterns and Finding Bugs in Quantum Programs. In *Proceedings of the 46th International Symposium on Computer Architecture.* ACM, 541–553.

[19] Nathan Killoran, Josh Izaac, Nicolás Quesada, Ville Bergholm, Matthew Amy, and Christian Weedbrook. 2019. Strawberry fields: A software platform for photonic quantum computing. *Quantum* 3 (2019), 129.

[20] David Layden, Sisi Zhou, Paola Cappellaro, and Liang Jiang. 2019. Ancilla-Free Quantum Error Correction Codes for Quantum Metrology. *Physical review letters* 122, 4 (2019).

[21] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems.* ACM, 1001–1014.

[22] Gushu Li, Yufei Ding, and Yuan Xie. 2020. Towards Efficient Superconducting Quantum Processor Architecture Design. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems.* 1031–1045.

[23] Ji Liu, Gregory T Byrd, and Huiyang Zhou. 2020. 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.* 1017–1030.

[24] Margaret Martonosi and Martin Roetteler. 2019. Next Steps in Quantum Computing: Computer Science's Role. *arXiv preprint arXiv:1903.10541* (2019).

[25] Mavadia et al. 2017. Prediction and Real-Time Compensation of Qubit Decoherence via Machine Learning. *Nature communications* 8 (2017).

[26] David C McKay, Thomas Alexander, Luciano Bello, Michael J Biercuk, Lev Bishop, Jiayin Chen, Jerry M Chow, Antonio D Córcoles, Daniel Egger, Stefan Filipp, et al. 2018. Qiskit Backend Specifications for OpenQASM and OpenPulse Experiments. *arXiv preprint arXiv:1809.03452* (2018).

[27] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. 2019. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems.* ACM, 1015–1029.

[28] Prakash Murali, David C McKay, Margaret Martonosi, and Ali Javadi-Abhari. 2020. Software Mitigation of Crosstalk on Noisy Intermediate-Scale Quantum Computers. (2020), 1003–1016.

[29] Daniel C Murphy and Kenneth R Brown. 2019. Controlling Error Orientation to Improve Quantum Algorithm Success Rates. *Physical Review A* 99, 3 (2019), 032318.

[30] Scott Pakin. 2016. A Quantum Macro Assembler. In *2016 IEEE High Performance Extreme Computing Conference (HPEC).* IEEE, 1–8.

[31] Scott Pakin and Steven P Reinhardt. 2018. A Survey of Programming Tools for D-Wave Quantum-Annealing Processors. In *International Conference on High Performance Computing.* Springer, 103–122.

[32] Tirthak Patel, Baolin Li, Rohan Basu Roy, and Devesh Tiwari. 2020. {UREQA}: Leveraging Operation-Aware Error Rates for Effective Quantum Circuit Mapping on NISQ-Era Quantum Computers. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20).* 705–711.

[33] John Preskill. 2018. Quantum Computing in the NISQ Era and Beyond. *Quantum* 2 (2018), 79.

[34] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I Schuster, Henry Hoffmann, and Frederic T Chong. 2019. Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems.* ACM, 1031–1044.

[35] Kaitlin N Smith and Mitchell A Thornton. 2019. A Quantum Computational Compiler and Design Tool for Technology-Specific Targets. In *Proceedings of the 46th International Symposium on Computer Architecture.* ACM, 579–588.

[36] Luyan Sun, Ling Hu, Yuwei Ma, Weizhou Cai, Xianghao Mu, Yuan Xu, Wang Weiting, Yukai Wu, Haiyan Wang, Yipu Song, et al. 2019. Experimental Quantum Error Correction with Binomial Bosonic Codes. In *APS Meeting Abstracts.*

[37] Swamit S Tannu and Moinuddin Qureshi. 2019. Ensemble of Diverse Mappings: Improving Reliability of Quantum Computers by Orchestrating Dissimilar Mistakes. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 253–265.

[38] Swamit S Tannu and Moinuddin K Qureshi. 2019. Mitigating Measurement Errors in Quantum Computers by Exploiting State-Dependent Bias. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 279–290.

[39] Swamit S Tannu and Moinuddin K Qureshi. 2019. Not All Aubits are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems.* ACM, 987–999.

[40] Barbara Terhal, Leonid Pryadko, Daniel Weigand, Yang Wang, Hamed Asasi, and Christophe Vuillot. 2019. Scalable Quantum Error Correction with the Bosonic GKP Code. In *APS Meeting Abstracts.*

[41] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. 2019. Mapping Quantum Circuits to IBM QX Architectures Using the Minimal Number of SWAP and H Operations. In *Proceedings of the 56th Annual Design Automation Conference 2019.* ACM, 142.

[42] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018).

[43] Alwin Zulehner and Robert Wille. 2019. Compiling SU (4) Quantum Circuits to IBM QX Architectures. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference.* ACM, 185–190.