EQC: Ensembled Quantum Computing for Variational Quantum Algorithms

Samuel Stein*, Nathan Wiebe◇*, Yufei Ding†, Peng Bo*, Karol Kowalski*, Nathan Baker*, James Ang*, Ang Li*
{samuel.stein,nathan.wiebe,peng398,karol.kowalski,nathan.baker,ang,ang.li}@pnnl.gov,yufeiding@cs.ucsb.edu
*Pacific Northwest National Laboratory, Richland, Washington, USA
†University of California, Santa Barbara, California, USA
◇University Of Toronto, Toronto, Ontario, Canada

ABSTRACT

Variational quantum algorithm (VQA), which is comprised of a classical optimizer and a parameterized quantum circuit, emerges as one of the most promising approaches for harvesting the power of quantum computers in the noisy intermediate scale quantum (NISQ) era. However, the deployment of VQAs on contemporary NISQ devices often faces considerable system and time-dependant noise and prohibitively slow training speeds. On the other hand, the expensive supporting resources and infrastructure make quantum computers extremely keen on high utilization.

In this paper, we propose a virtualized way of building up a quantum backend for variational quantum algorithms: rather than relying on a single physical device which tends to introduce ever-changing device-specific noise with less reliable performance as time-since-calibration grows, we propose to constitute a quantum ensemble, which dynamically distributes quantum tasks asynchronously across a set of physical devices, and adjusts the ensemble configuration with respect to machine status. In addition to reduced machine-dependant noise, the ensemble can provide significant speedups for VQA training. With this idea, we build a novel VQA training framework called EQC - a distributed gradient-based processor-performance-aware optimization system - that comprises: (i) a system architecture for asynchronous parallel VQA cooperative training; (ii) an analytical model for assessing the quality of a circuit output concerning its architecture, transpilation, and runtime conditions; (iii) a weighting mechanism to adjust the quantum ensemble's computational contribution according to the systems' current performance. Evaluations comprising 500K times' circuit evaluations across 10 IBMQ NISQ devices using a VQE and a QAOA applications demonstrate that EQC can attain error rates very close to the most performant device of the ensemble, while boosting the training speed by 10.5× on average (up to 86× and at least 5.2×). EQC is available at https://github.com/pnnl/eqc .

CCS CONCEPTS

• **Computer systems organization → Quantum computing**.

ISCA '22, June 18–22, 2022, New York, NY, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8610-4/22/06.
https://doi.org/10.1145/3470496.3527434

KEYWORDS

Quantum Computing, Variational Quantum Algorithms, Distributed Computing

ACM Reference Format:
Samuel Stein*, Nathan Wiebe◇*, Yufei Ding†, Peng Bo*, Karol Kowalski*,, Nathan Baker*, James Ang*, Ang Li* . 2022. EQC: Ensembled Quantum Computing for Variational Quantum Algorithms. In *The 49th Annual International Symposium on Computer Architecture (ISCA '22), June 18–22, 2022, New York, NY, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3470496.3527434

1 INTRODUCTION

Quantum computing (QC) is poised to offer substantial computational capabilities that classical computing could never feasibly reach in many critical domains, such as database search [20], graph combinatorial optimization [16], quantum chemistry [7], machine learning [4], etc. Recently, Google has showcased quantum supremacy on their 53 qubits *Sycamore Quantum Processing Units* (QPU), where a quantum sampling problem, which would take an estimate of 10,000 years (rectified to 2.5 days by IBM) running on the ORNL Summit supercomputer now can be finished in ∼200 seconds [1].

Although many algorithms that could provide quantum supremacy have been theorised [20, 44], QPUs in the contemporary NISQ era are susceptible to noise in a multitude of ways. These include (i) *Coherence Error*, which describes the natural decay of qubit states to their ground states, analogous to retention error in classical devices. Coherence error includes noise-induced state decay (i.e., T1 decay) and spin-spin relaxation (i.e., T2 decay); (ii) *Gate Error*, which refers to imperfect gate operations. Such gate error is relatively small for 1-qubit gates (e.g., ∼0.1% for IBMQ), but can be significant for 2-qubit gates (e.g., nearly 4% for IBMQ [51]); (iii) *SPAM Error*, short for state preparation and measurement error, is introduced due to unsatisfactory state initialization and measurement sensing. Additionally, non-coherent interference such as cross-talk [31, 43] can impose computational error, acting as device-specific latent noise.

Due to the high error rate and insufficient qubits in supporting full-scale *quantum error correction* (QEC), people are searching for quantum algorithms that can harvest the quantum advantages of NISQ devices but are more robust to the errors. The *Variational Quantum Algorithm* [7, 10, 15] stands out as one of the most promising NISQ algorithms, which employs a classical optimizer to train a parameterized quantum circuit.

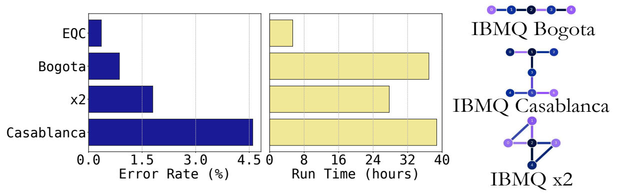

Figure 1: VQE error rate, running time and quantum processor topologies on IBMQ.

VQAs have been applied for several key applications. These include (i) the *Variational Quantum Eigensolver* (VQE) for investigating molecular and nuclear structures/dynamics in quantum chemistry [2, 54] and nuclear physics [26, 40]; (ii) *Quantum Approximate Optimization Algorithm* (QAOA) for optimizations such as MaxCut [11] in graph analytics; and (iii) *Quantum Neural Networks* (QNNs) for quantum machine learning [16, 22, 37, 45, 46].

Although VQAs have been extensively evaluated on platforms such as IBMQ [8], the deployment of VQAs for practical domain utilization (e.g., estimating the ground energy for molecules) on the NISQ platforms is still hampered by three challenges: (i) *The training is susceptible to device-specific bias and noise.* Similar to the correlated error caused by mapping a circuit to the same qubits and links repeatedly following a fixed pattern [51], we have observed that QC is also subject to device-specific bias originating from topology, SPAM [48], calibration and running conditions. Algorithms such as Quantum Classifiers or Quantum Generative models [22, 46] require generalised learning, and need general models for later reinduction. Although certain VQAs do not need to repeat induction, this learned-redundancy requires the VQA to complete training between calibration times on one machine. Figure 1-left shows the error rates of training a VQE circuit on three IBMQ devices (`Bogota`, `x2`, and `Casablanca`) individually with respect to ideal simulation, as well as the error reduction that can be achieved through our EQC framework using solely these same three machines in an ensemble; (ii) *Prohibitively long execution time.* Due to the significant deployment, maintenance, and operating costs, most QC platforms are provided as a cloud service and shared by many users. Given that VQA needs to repeatedly test & adjust the parameterized circuit, iterate over enormous input data, train over numerous epochs, and wait for each trial going through the waiting queue, this leads to extremely long execution time. For example, it takes us about 4,623 hours to train a VQE circuit on Manhattan. The congestion, calibration and maintenance can further exacerbate the delay. Figure 1-middle shows the running time for our VQE circuit on the three IBMQ devices, compared to our ensemble approach; (iii) *Large utilization variance due to unbalanced workloads.* Despite the sharing nature, quantum computers can be underutilized. This is largely due to workload imbalance across multiple devices given user-preference and congestion. As users tend to select the best performing quantum computer, the overall provider processor utilisation can be quite unbalanced. The VQA jobs significantly amplify such imbalance due to iterative testing and long execution time. Considering the expensive cost from the supporting infrastructure and sources such as cryogenic coolers, superconducting wires, microwave devices, and sophisticated control circuitry [14], the under-utilization can be a considerable burden to the QC service providers. State of the art technologies such as Qiskit Runtime improve VQA performance through minimizing classical-quantum communication costs. However, these vertical approaches do not help with machine-specific bias, and may further exacerbate the unbalanced utilization problem.

In this paper, we propose EQC – an Ensembled Quantum Computing framework for Variational Quantum Algorithms. EQC can bring considerable stability, noise-reductions and speed-ups in VQA training over alternative approaches. Being the primary effort in system-managed QPU-aware parallel VQA training, EQC employs a master node to asynchronously manage a set of client nodes paired with backend devices, comprising the quantum ensemble and performing asynchronous gradient descent. To dampen the QPU bound noise and minimize machine-bound noise, EQC further adopts an analytic model to assess the quality of the quantum processor, and builds a management module to allow online adjustment of the quantum ensemble based on the runtime condition of the backend devices. Evaluations on 10 IBMQ devices show that EQC can mitigate system-dependant bias while boosting the speed by over 10× (up to 86×). This paper thus makes the following contributions:

- ***The Concept and Design of Quantum Ensemble:*** we propose a novel way of thinking about a quantum backend: rather than using a single physical device incorporating device-specific bias and slow execution time, a quantum ensemble can serve as a virtualized backend for offering adaptive bias and noise-mitigation through calibration-aware device mixtures, improved device-choice relating to circuit design, and faster execution through parallelization.
- ***The EQC Framework:*** we develop a system architecture for asynchronous VQA optimization that can bring over an order of speedups over single QPU-based optimization, whilst simultaneously reducing error. We provide theoretic proof on the convergence of the approach.
- ***The Adaptive Weighting System:*** we propose a weighting approach considering noise level, topological constraint, transpiled circuit structure, calibration time, etc. It thus allows adaptive and dynamic weighting of the gradient with respect to the device condition, mitigating time-dependent drift and overwhelming time-dependant machine bias.

2 BACKGROUND

2.1 Quantum Gates and Devices

QC exploits the quantum phenomena such as superposition and entanglement for computing, and manipulates the probabilistic state-vector to accomplish tasks. State-vectors, which represent the quantum states, are linear combinations of orthonormal eigenvectors with normalized square sum coefficients. Quantum computers make use of gates to perform state transformations, represented as unitary matrices relative to some basis, which operate on the state vector. Gate operations are performed in a physical quantum processing unit (i.e., QPU). There are two main features for QPUs:

(i) Basis gates. Basis gates describe the native hardware operations; all quantum circuits need to be transpiled to basis gates eventually in order to be executed by a QPU. For example, most IBMQ devices use the same basis gate set: `CNOT`, `ID`, `RZ`, `SX`, `X` [13].

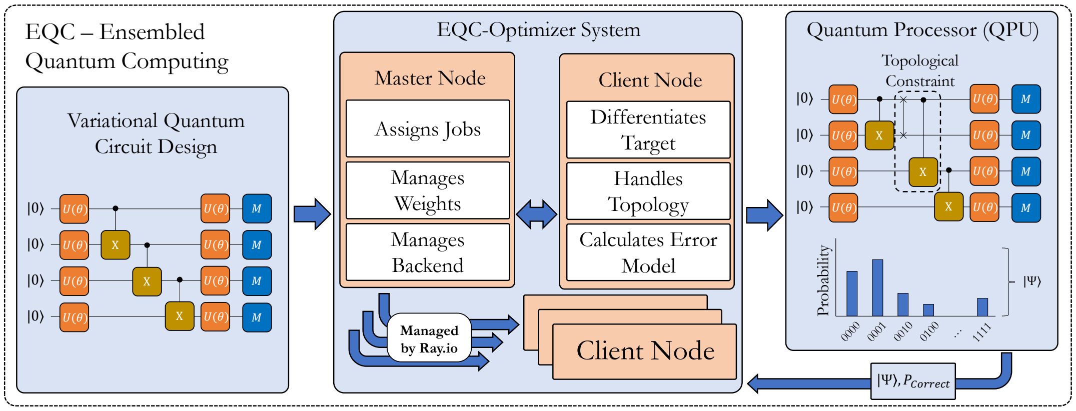

Figure 2: Overview of EQC framework.

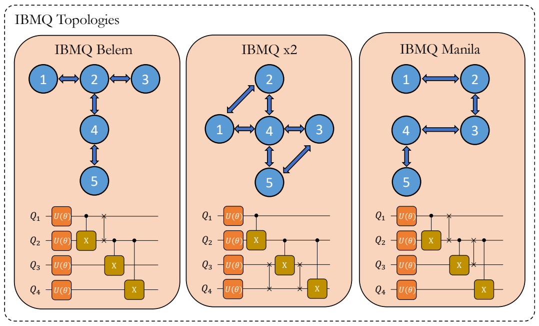

Figure 3: Topology illustration of 3 IBMQ quantum machines. Bi-directional connections indicate physical connections between qubits. Circuits illustrated perform the same computation transpiled to the respective topological constraints.

(ii) Topology. Applying two-qubit gates such as `CNOT` in theoretical development usually assumes fully connected systems. However, on real superconducting quantum processors, due to limitations on the physical designs, inter-qubit connectivity constraints exist.

As shown in Figure 3, the three devices (IBMQ `Belem`, `x2` and `Manila`) all have 5 physical qubits. However, due to the variance of inter-qubit connectivity, the circuit structures for the same function can be different after transpilation. This is because the two-qubit gates such as `CNOT`, can only be performed on connected qubits; otherwise, the qubits must be moved next to each other using SWAP-gates. The swap gate is composed of three CNOT gates, a costly operation. The SWAP-gates allow any two qubits that are connected to swap positions, thus can hide the topology constraint, however at a substantial time and compounding error cost.

2.2 QPU Noise

QPU Noise: NISQ QPUs are highly diverse in their ability to run quantum routines without compounding a multitude of random errors due to low-level qubit stability and high-level QPU topological constraints. As discussed, there are three types of errors: (i) *Coherence Error* due to the non-ideal energy exchange between the outside system and the ideal system, which is characterized by exponential decay factors $T1$ and $T2$, and known as thermal dephasing and decoherance [17]; (ii) *Gate Error* due to depolarization, which is characterized by undesired Pauli-X/Y/Z operations representing bit-flips or phase-flips. (iii) *SPAM Error* due to state preparation and measurement noise, which is associated with the anharmonicity between the states $|0\rangle$ and $|1\rangle$, and the erroneous process of distinctively and properly distinguishing between the two states correctly.

Additionally, each QPU has its own unique noise profile that changes with frequent calibration. These volatile systems vary in spatial and temporal noise due to the imperfect manufacturing process [31], imperfections of gate implementation and control, and specific external interference [30, 33]. These noise compound with each other, increasing the overall probability of obtaining an erroneous outcome.

2.3 Variational Quantum Algorithm

VQAs are characterized by a group of parameterized quantum operations, with parameters $[\vec{\theta}]$, $\theta \subseteq R$. The quantum algorithms goal is to optimize $[\vec{\theta}]$ to $[\vec{\theta}^*]$, where $[\vec{\theta}^*]$ is the optimal learned parameters, such that the trial wave function of $|\psi\rangle$ is transformed closer to some target wave function according to an optimization function. The optimization function represents for example the ground state of a quantum system, and is expressed as parameterized unitaries $U(\theta_i)U(\theta_{i-1})...U(\theta_1)|\psi\rangle$. For *Variational Quantum Eigensolvers* (VQE) [37] or *Quantum Approximate Optimization Algorithm* (QAOA) [16], the objective is to optimize towards some Hamiltonian H describing the total energy of the system, or representing some objective cost function, as shown in Equation 1:

$$argmin([\vec{\theta}])\ \langle\psi([\vec{\theta}])|\,\mathrm{H}\,|\psi([\vec{\theta}])\rangle \tag{1}$$

This optimization process is tailored for alternative use cases typically in quantum machine learning [22, 42, 45, 46].

The present VQA optimization process, however, encounters two key issues: (i) There is a substantial computational cost in inducing circuits and optimizing each individual parameter; (ii) Machine noise leads to a bias being learnt by a machine during training, where the parameters will deviate compensating for a machines noise. However, when a machine is re-calibrated or the backend device is changed, the learned parameters are no longer appropriate. For example, variational quantum machine learning models such as classifiers or generative models require to be general in order to be seen a good solution [9]. Furthermore, if the noise is overwhelmingly high on a machine, the gradient computation will be erroneous and the algorithm will not learn.

2.4 Distributed Optimization

Distributed optimization of classical deep neural networks (DNNs) has been motivated by the need to train large neural networks [6, 24] for exceedingly large datasets [27] through parallel processing. Asynchronous stochastic gradient descent (ASGD) has been shown to provide substantial speedups for DNN training with little accuracy loss [55]. However, the problem of distributing optimization across multiple classical processing units changes substantially when looking at a quantum distributed system. Classical systems primarily deal with latency, whereas quantum processors deal with both latency and individual noise characterization of each QPUs [21]. The problem is exacerbated when multiple backend device architectures and/or providers are presented. In addition to the different runtimes of the service, the distinct periods of calibration which are poised to become more frequent [52], the diverse range of speed, as well as the noise can make the coordination of the training process significantly more complicated than classical distributive DNN training.

3 EQC SYSTEM DESIGN

In this section we describe the EQC architecture, structure and discuss its implementation.

3.1 VQA Task Decomposition

VQA optimization typically adopts stochastic gradient descent to perform step by step parameter differentiation, where concurrent tasks can be extracted from multiple hierarchies of the algorithm for (asynchronously) parallel execution.

Regarding **VQE**, these problems are commonly related to quantum chemistry [23, 34, 53], involving a Fermionic Hamiltonian that is decomposed via processes such as Jordan-Wigner decomposition. As a decomposed Hamiltonian is a linear sum of Pauli strings generated by transforming spin operators into a series of quantum computing supported Pauli operators. In this case, we perform the system parallelization at the Pauli string level. To compute the derivative of each respective parameter, the system collects parallelized computation, to perform one gradient descent step on one parameter. This is repeated for each parameter in the VQE ansatz.

Regarding **QAOA** [47], comprising of optimization problems such as MaxCut, the learning process aims to optimize over a target Hamiltonian objective. In the case of a single matrix representation of a Hamiltonian, the system aims to minimize or maximise $\langle\psi|H|\psi\rangle$. In doing so, we parallelize at the parameter level, where the derivative for each parameter is parallelized. This results in each parameter being asynchronously optimized, and is repeated over all parameters. Each parallel task is responsible for computing one gradient descent step over one parameter.

Regarding **QNN** [4, 22, 41, 46], the learning process involves optimizing over $\frac{\delta f(x,\theta)}{\delta\theta} = \frac{1}{n}\sum_{i=1}^{n}\frac{\delta f(x_i,\theta)}{d\theta}$, where the average cost over a dataset is to be minimized. To parallelize the QNN training, we distribute at the dataset level for parameter optimization. Given a data set, as the parameter gradient is the average of all the data points' derivative for a parameter, the gradient can be parallelized for each parameter at each data point. Each parallel job is responsible for computing the gradient with respect to its assigned data point for a target parameter. A complete gradient descent step on one parameter thus is comprised of the average returned gradient for the target parameter over all data points, despite the gradients are applied asynchronously.

3.2 EQC System Architecture

As shown in Figure 2, EQC is built as a single-master node, multi-client nodes architecture. It is designed for managing and optimizing the training of a VQA circuit with a set of parameters to optimize using a set of NISQ QPUs with different noise profiles, different topologies and different induced circuit architectures that may vary with time and transpilation. EQC consists of the master node, the client nodes, and the QPUs: the master node distributes workloads to client nodes, which perform target derivative calculation with their paired QPUs for their assigned tasks. Computing a derivative on a NISQ-term QPU is challenged by the noise factors, resulting in erroneous gradient computation. Therefore, the client node also computes a weight based on the transpiled circuit and reported system noise statistics at the circuit induction time as well. The utilization of asynchronous stochastic gradient descent across a set of NISQ QPUs, whilst incorporating a QPU-aware weighting system ensures the EQC system attains a computational speedup, and improves system optimization performance.

3.3 EQC Implementation

We present the design details about the three major components of EQC: *master node*, *client node*, and *QPU*.

3.3.1 Master Node. The master node is provided with three key pieces of information: (i) the quantum circuit over which the system is to be optimized with measurements; (ii) the initialized parameters $[\vec{\theta}]$, $\theta \subseteq R$; and (iii) a detailed loss function ℓ describing the optimization problem.

The master node is responsible for keeping track of all parameters at all times, as well as the ideal (i.e. no topological constraints) circuit layout. Once the master node has been initialized with all of the pre-requisites in place, the master node queries the quantum computing service provider(s) based on the ensemble configuration. The available QPUs to form the ensemble should have active qubits larger than the number of qubits required by the parameterized circuit. The backend devices can be homogeneous or heterogeneous. In other words, although we demonstrate EQC using 10 of IBMQ devices, the framework itself supports an ensemble of QPUs from different vendors.

When the master node dictates the formalization of the ensemble, it initializes client nodes one per device with the loss function and circuit template. The master node then distributes the VQA task parameters in a cyclic fashion to the client nodes. The master node adopts a task queue for parallel task distribution. If new QPUs become available, the master node initializes client nodes as needed and assigns the nodes the proceeding jobs. The master node iterates the parameter list and asynchronously, i.e. no point of synchronisation, assigns the next parameter to a client node that

Algorithm 1 EQC master node

```
Set $\epsilon_{\text{END}}$, $\alpha$ and $\ell$
$C \leftarrow$ Circuit Template
Parameters $\leftarrow [\vec{\theta}]$
$\epsilon, i \leftarrow 0$
$K \leftarrow [QPUs]$
$G \leftarrow$ [k Client Nodes]
for QPU, Client in $K, G$ do
    $Client \leftarrow QPU, C,$ and $\ell([\vec{\theta}])$
end for
for Param in Parameters do
    $G_i \leftarrow$ Compute $\frac{\delta\ell}{\delta\theta_i}$
    $i \leftarrow i + 1$
    if $i > len([\vec{\theta}])$ then
        $i \leftarrow 0$
    end if
end for
while $\epsilon < \epsilon_{END}$ do
    for CN in Ready Clients do
        $\frac{\delta\ell}{\delta\theta_i}, P_{\text{Correct}} \leftarrow CN$
        $P_{\text{Correct}} \leftarrow Bound(P_{\text{Correct}})$
        $\theta_i = \theta_i - P_{\text{Correct}}(\alpha\frac{\delta\ell}{\delta\theta_i})$
        $CN \leftarrow [\vec{\theta}]_{\text{NEW}}$
        $CN \leftarrow$ Calculate $\frac{\delta\ell}{\delta\theta_i}$
        $i \leftarrow i + 1$
        if $i > len([\vec{\theta}])$ then
            $i \leftarrow 0$
            $\epsilon \leftarrow \epsilon + 1$
        end if
    end for
end while
```

is available until no parameters left. After that, the next epoch can start.

The only information the master node receives from the client nodes are the gradients of the parameters θ_i, and the computed noise model of the QPU associated, which we will discuss in the following section. On receipt of a gradient, the master node applies this gradient using the ASGD rule, and sends a new parameter to differentiate at an idle client. The whole process is outlined in Algorithm 1.

3.3.2 Client Node. The primary objective for the client nodes is to maximize the usage time of each QPU available. Each client node is responsible for managing one QPUs experiments and maximizing its utilization. On initialization, each client node is provided with the quantum circuit template, the unique id, and the optimization function ℓ.

The client node then receives a parameter index to differentiate and generates the forward and backward pass from the parameter shift rule. The client node has the information about the topological constraints of the resident QPU, as well as all QPU noise factors such as T1, T2 and CNOT error rates. Using the topological constraints, the client node transpiles the template circuit with respect to the target QPU topology. The client node runs the transpiled forward and backward circuit in the QPU with sufficient shots to generate a

Algorithm 2 EQC client node

```
$C \leftarrow$ Circuit Template
$Q \leftarrow$ Connect to QPU
$C_{\text{Transpiled}} \leftarrow Transpile(C, Q)$
$[\vec{\theta}] \leftarrow$ Parameters
while Calculate $\frac{\delta\ell}{\delta\theta_i}$ do
    $[\vec{\theta}] \leftarrow [\vec{\theta}]_{\text{NEW}}$
    $[\vec{\theta}]_{\text{FWD,BCK}} \leftarrow [\vec{\theta}], [\vec{\theta}]$
    $[\vec{\theta}]_{\text{FWD,BCK}} \leftarrow [\vec{\theta}_i] \pm \frac{\pi}{2}$
    Job $\leftarrow$ Submit $C_{Transpiled}([\vec{\theta}])_{\text{FWD,BCK}}$
    $P \leftarrow P_{\text{Correct}}(C_{\text{Transpiled}}, QPU)$
    if Job is Ready then
        $|\Psi\rangle_{\text{FWD,BCK}} \leftarrow$ Job Results
        $\frac{\delta\ell}{\delta\theta_i} = \frac{\ell(|\Psi\rangle_{\text{FWD}} - \ell(|\Psi\rangle_{\text{BCK}}}{2}$
        Ready$(\frac{\delta\ell}{\delta\theta_i}, P_{\text{Correct}})$
    end if
end while
```

confident forward and backward pass probability distribution, and processes these two distributions attaining $\frac{\delta\ell}{\delta\theta_i}$.

Finally, the client node returns the obtained gradient and an associated noise factor back to the master node, and in return acquires the next pending job to repeat this process. The procedure of the client node is shown in Algorithm 2.

3.3.3 QPUs. Each quantum processor has its own topological constraints, architecture type and noise factors (see Table 1). The topology can be represented by a non-directed graph, as shown in Figure 3. To accommodate such topological constraints, extra SWAP operations are needed. With respect to noise, each QPU is substantially different. Highly connected QPUs suffer from cross-talk, degraded CNOT-gate fidelity and T1/T2 lifespans. As a comparison, newer architectures such as the IBM Falcon processor are laid out in a honeycomb fashion to mitigate the cross-talk issue and reduce noise at the cost of a lower connectivity. Nevertheless, noise still applies to each NISQ system, and is correlated to the topological constraints considering SWAPs and circuit depth.

3.4 Implementation Details

EQC is implemented based on two packages: *Qiskit* and *Ray*. Qiskit is responsible for circuit generation, communication management with IBMQ devices, and all other quantum related processes. Meanwhile, Ray [29] provides very flexible and universal APIs for building distributed systems. Ray was originally designed for distributive reinforcement learning. Ray.io creates remote actors generated based on Python's class structure, which are responsible for managing the QPU-related configuration information, and properties mentioned above with respect to client node responsibilities. Ray.io allows for asynchronous communication between the master node and client nodes, through the use of "ready" calls that indicate the client node has a pending object to be collected. Each client node connects to one running, unreserved QPU by connecting to individual IBMQ backend.

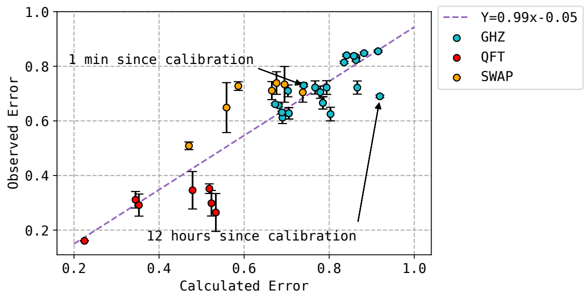

Figure 4: Calculated vs observed 5-Qubit GHZ state error. Observed error indicates proportion of erroneous outputs, and calculated error indicates the predicted error chance.

4 WORKER QPU WEIGHTING

NISQ devices are extremely noisy and they compound computational errors in a multitude of ways. Furthermore, their noise models change at every calibration. Certain quantum computers are better at producing more accurate resultant probability distributions than others, due to different coherent and latent system noise factors. In order to take these noise factors into consideration for EQC, we propose to quantify and weight the gradient based on an analytic model.

A key observation that aligns system noise (measurement, depolarization, and thermal/dephasing decay) here is that they can be attributed as probabilistic events. Each quantum computer, when calibrated, reports the gate fidelity, measurement fidelity, gate times, state anharmonicity, and T1/T2 decay constants. The event of the identity gate being applied (i.e. no error) is $1 - p$ for each gate fidelity reported. For thermal decay, the exponential mode of $P(\Phi_{x,t} = \Phi_{0,t}) = e^{-t/T1}$ where $\Phi_{x,t}$ represents the probability of a decay occurring between time 0 and t. The probability of a error-free computation occurring thus can be expressed as the product of possible errors not occurring. Therefore, we propose characterizing an entire systems quality by compounding these calibration statistics based probabilistic models with quantum routine properties described below:

$$P_{Correct} = e^{\frac{-CD\frac{\mu_{t-G1}+\mu_{t-G2}}{2}}{T_1T_2}}(1-\gamma)^{G_1}(1-\beta)^{G_2}(1-\omega)^M \quad (2)$$

where CD refers to the critical depth of a circuit, $\mu_{t-G_{1/2}}$ is the average gate time of a 1 or 2 gate operator, β is the CNOT fidelity, γ is single gate fidelity, $G_{1/2}$ is the number of single or dual gate operations, ω is the measurement error rate and M is the number of measurements. Since this equation is a probabilistic model, it is bound by $0 \leq P_{Correct} \leq 1$. Furthermore, our model benefits from being topology-aware. By incorporating dual gate count G_2, topological constraints will drive this value down due to increased SWAP gates, increasing the value of G_2, thereby decreasing weights of computationally consequential topologies.

To validate our analytic model, we apply a 5-qubit GHZ state, a 3-qubit QFT+Inverse QFT, and a 5-qubit SWAP Test between two identical quantum states onto real quantum computers and use our model to predict the probability of error. All of these algorithms have an expected bit string output, and hence any incorrect output is treated as an incorrect run. We compute our expected $P_{Correct}$ and compare with IBMQ's QPUs: `Lima`, `x2`, `Casablanca`, `Toronto`, `Belem`, `Quito`, `Manila` and `Bogota` in Figure 4.

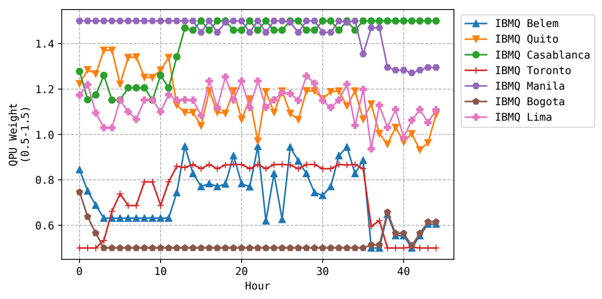

Figure 5: QPU weighting. Weighting system is applied with weights bound in the range $[0.5, 1.5]$ **over 7 QPUs transpiling and computing Equation 2 over circuit illustrated in Figure 8.**

As can be seen in Figure 4, a strong linear positive correlation is demonstrated between observed errors and $P_{Correct}$ with a R^2 value of 0.597. Furthermore, we calculate the Pearson correlation between our calculated errors and observed errors. A correlation of 0.888 is attained with a two-tailed p-value of 9.27E-15, indicating that our results and actual error rates are strongly statistically correlated. We observe that our model was substantially better at predicting the chance of an error closer to calibration times. As observed in Figure 4, for a freshly calibrated system we observed an error rate of 73.5%, alongside a calculated error rate of 74.0%. However, for staler calibrations we observe an error rate of 69.0% alongside a calculated error rate of 91.9%. This was only observed on certain machines however, with other machines not suffering from the same problem of stale calibrations. We make use of our characterizing formula by giving each worker node the job of calculating its $P_{Correct}$ at transpilation before each circuit induction. We utilize our model by linearly scaling the distribution of $P_{Correct}$ values to be bound by a range (see Figure 5). This weighting system is real-time adaptive, and incorporates the transpiled cost of a circuit and the QPUs stability. We demonstrate this in Figure 5 where a group of IBMQ machines and client nodes are bound to have weights between $[0.5, 1.5]$. The circuit transpiled is outlined in Figure 8. We demonstrate the reported weights of an ensemble of 7 machines over a 48 hour time period, and how the weights reported change with time. The determination of a QPU weight is performed prior to circuit induction, based on the transpiled circuit structure. Once the jobs are completed, the QPU weight is sent back to the master node along side the gradient. These values are combined in the update rule shown in Equation 4.

5 EQC EVALUATION

5.1 VQA Problem Settings

We comprehensively evaluate EQC using the Variational Quantum Eigensolver optimization problem, as well as the Quantum Approximate Optimization Algorithm. The experimental setup is illustrated in Figure 7. We use up to 10 IBMQ machines in deploying and evaluating our system, listed in Table 1. In total, our evaluations for this work comprise ~500,000 circuit executions on IBMQ devices.

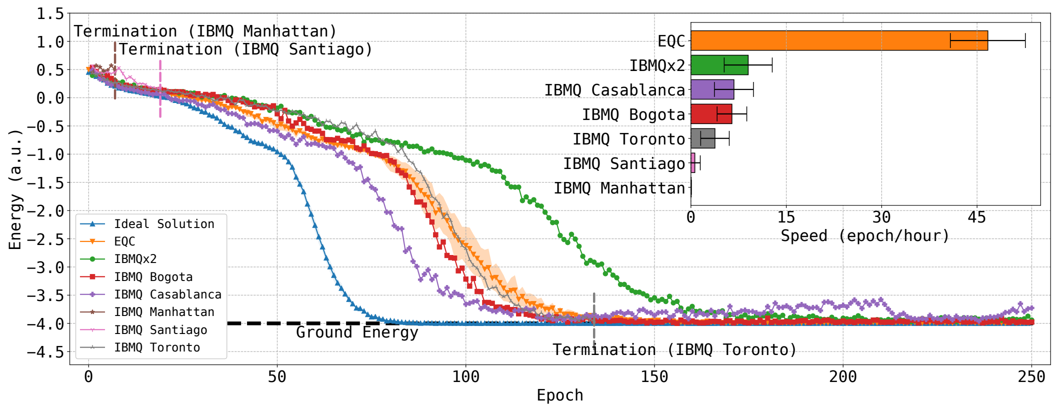

Figure 6: 4-Qubit Heisenberg model on a square lattice. Termination represents an experiment being terminated beyond 2-weeks of running time. Error bars are present for EQC and the ideal solution. Independent machine training is represented by a machine name in the legend.

Table 1: IBMQ platforms used for evaluation. QV refers to quantum volume [12]. The line, T-shape and fully-connected topology can be seen in Figure 3.

Device	Qubits	Processor	QV	Topology
Lima	5	Falcon r4T	8	T-shape
x2	5	Falcon r4T	8	Fully-connected
Belem	5	Falcon r4T	16	T-shape
Quito	5	Falcon r4T	16	T-shape
Manila	5	Falcon r5.11L	32	Line
Santiago	5	Falcon r4L	16	Line
Bogota	5	Falcon r4L	32	Line
Lagos	7	Falcon r5.11H	32	H-shape
Casablanca	7	Falcon r4H	32	H-shape
Toronto	27	Falcon r4	32	Honeycomb
Manhattan	65	Falcon r4	32	Honeycomb

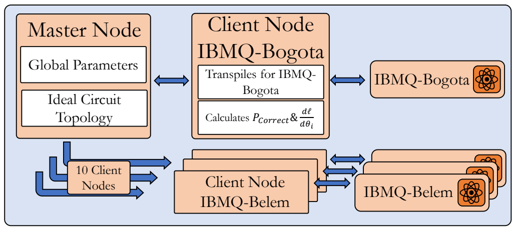

Figure 7: Experimental setup.

Both problem settings are 4 qubit problems, due to the majority of NISQ era devices being limited in size.

5.2 VQE Evaluation

The target problem is an application to quantum magnetism, whereby we perform the minimization of a 4-qubit Heisenberg model Hamiltonian on a square lattice [23]. The Hamiltonian is described by:

$$H = J \sum_{(i,j)} (X_i X_j + Y_i Y_j + Z_i Z_j) + B \sum_i Z_i \tag{3}$$

where J refers to the spin-spin strength and B is the magnetic field along the Z-axis. We evaluate the Hamiltonian that has $J/B = J = B = 1$, the same experiment as already demonstrated in [23], which provides a reliable baseline (i.e., ground energy) for

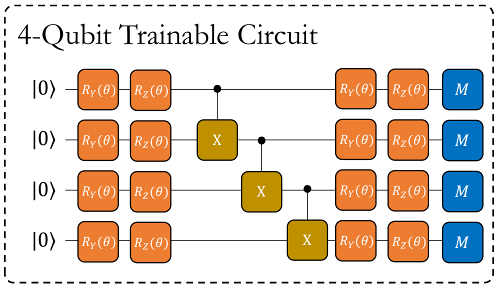

Figure 8: 4-Qubit VQE circuit. X gates connected to another qubit represent CNOT gates. M gates represent measurements.

verifying the convergence. The 4-qubit square lattice results can be represented as a graph with $V = [1, 2, 3, 4]$ and edges $E = [(1, 2), (2, 3), (3, 4), (1, 4)]$. We make use of a hardware-efficient ansatz comprised of a linearly entangled full bloch-sphere rotation style circuit. The parameterized circuit is visualized in Figure 8, composed of R_Y, R_Z and $CNOT$ gates. This circuit design exploits a full bloch-sphere rotation in the initial rotation. This is followed by each qubit being linearly entangled with one and other (i.e. CNOT(1,2), CNOT(2,3), ... (CNOT(n-1,n)), and then followed by the rotation layer. This is similarly motivated to the circuit design used in [23]. We make use of parameterized differentiation according to the ASGD update rule at a learning rate (α) of 0.1. To incorporate node weighting, we make use of Equation 4.

$$\theta_i^{t+1} = \theta_i^t - P_{\text{Correct}} \alpha g^\tau (\theta_i^\tau) \tag{4}$$

5.3 VQE Unweighted Performance Evaluation

We show how EQC can contribute to noise-mitigation and machine-bias mitigation, while gaining significant performance improvement over single-machine optimization. The primary results are shown in Figure 6. We show the optimization process of minimizing the system energy as described in Equation 3 for over 250 epochs. The ideal learnt ground state is illustrated at −4.0 a.u. (as validated in [23]). We further illustrate the average single machine speed in comparison to EQC. We evaluate the optimization of our system on an ideal quantum simulator using 8192 shots with an average convergence point of 80 epochs over 10 experiments. This serves as the baseline as plotted in Figure 6. Notably, the standard deviation is plotted, however the value is negligible and hence is difficult to see on the plot. We further run the optimization problem once over 6 single IBMQ machines, each attempting to train the system on their own, independent of each other. Note, the optimization task is run only once on the single machines due to severe time costs of single-machine VQE training.

Overall, `IBMQ x2` is shown to be the slowest at converging with an approximate convergence at 175 epochs, a 118.75% increase in the number of epochs to converge. The relatively slow convergence is attributed to IBMQ `x2`'s older topological architecture and high degree of crosstalk, therefore being substantially more error prone. In comparison, for IBMQ `Bogota`, we observe convergence substantially quicker at epoch-122, 52.5% slower than an ideal simulator. Certain experiments were infeasible to complete, e.g., for IBMQ `Manhattan`, running a 250-epoch 16-parameter VQE requires at the order of months to train. With the present queuing time, it is expected it will take 193 days to reach the 250^{th} epoch. Similarly, `Santiago` would be on the order of weeks, with our approximation being 21 days. These numbers are also susceptible to large time-dependent fluctuations, with `Toronto` fluctuating from 6.553 epochs/hour to 0.033 epochs/hour over our entire training period.

Due to these large swings in performance, we terminate certain experiments which had not finished after 2 weeks of training, as seen in Figure 6. IBMQ `Casablanca` was substantially more performant at an average performance of 6.767 epochs/hour, hence only a run time of 36.944 hours, and converging at epoch 130. Although `Casablanca` was most performant initially in the epochs between 0 and 130, we observe the machine becoming unstable until epoch 215 where it stabilizes back at the ground energy. This is an example of time dependent drift whereby the machine becomes noisier over time, and the learned parameters are biased, compensating for this noise, and avoiding the actual optimal solution.

To summarize, two key factors are observed in the single-machine training evaluation: *machine throughput* and *machine stability*. Both of them are highly variable with time.

We evaluate the same problem on EQC, as illustrated in Figure 6. We observe an average speed of 46.701 epochs/hour when using our system over 10 IBMQ machines. The fastest machine is `x2`, performing at a rate of 9.011 epochs/hour. With EQC, the worst-case improvement of training speed is 518%. We train our system through EQC for 3 times and illustrate the standard deviation at each epoch in Figure 6. EQC converges at epoch 135, being 69% slower than an ideal simulator. This is substantially better than the noisiest but is also the fastest machine IBMQ `x2`. In fact, `x2` contributes the largest throughput to EQC. The key observation is that we demonstrate noise mitigation and machine-bias mitigation through EQC. With EQC being an ensemble of QPUs each with their own machine-specific noise profile, the problem of large machine-induced bias is dampened inherently through the mixture, i.e., an ensemble of averaged noise.

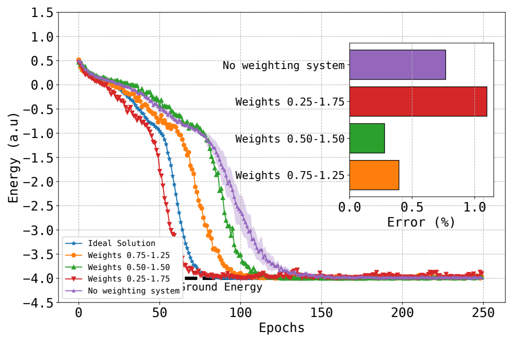

Figure 9: 4-Qubit Heisenberg weighted QPU results. Results from EQC are evaluated using the update rule in eqn-4, where no weights equates to $w = 1$. Error (%) represents average error rate relative to ground energy of -4.0 a.u.

Quantum processors tend to have time dependent drift from the ideal parameters. For example, `Casablanca` converges the fastest, however drifts from the ideal solution after convergence, as shown in Figure 6. This dampening is demonstrated by the final average converged energy of each system. With an ideal ground energy of −4.0, the error of a system is the average of the obtained ground energy divided by the ideal ground energy. The deviation from ground state error is visualized in Figure 1-right. As can be seen, EQC attains an error rate of 0.379%, whereas IBMQ `x2` attains an error rate of 1.798%. `Bogota` attains 0.865%, and `Casablanca` attains 4.6%. We can see that EQC achieves the closest result to the ideal solution when compared with training on single machines, implying that the ensemble can mitigate device-specific noise.

5.4 VQE Client Node Weighting Evaluation

We evaluate EQC against two different weighting strategies, and compare them to the baseline (i.e., without weighting) and the ideal simulation result. As shown in Figure 9, the application of our weighting model allows for an improvement in the rate of convergence. The weighting system operates by taking the maximum $P_{Correct}$, and the minimum $P_{Correct}$ and linearly normalizing the values to the weight bound range. As a concrete example, to weight a system of values between 0.5 − 1.5, the $P_{Correct}$ values over all client nodes are normalized and shifted +0.5. In the case of a modest scaling factor between 0.5 − 1.5, we observe a convergence at 115 epochs, against the regular 140 epochs, resulting in a 17.9% speedup. In the case of 0.25 − 1.75, the speedup is even further improved, converging at the same time as the ideal solution at 80 epochs, a 75% speedup. Due to the non-Gaussian distribution of weights, higher weights can correspond to better-than-ideal rate of convergence. In evaluating 0.75−1.25, we notice an improvement in convergence when comparing with 0.5 − 1.5 and no weights, which converges at 105 epochs. With respect to converged error rate, we plot this information within the sub figure of Figure 9. As illustrated in Figure 9, utilizing a weighting system of 0.25-1.75 increases error by 0.33%. However, utilizing weights of 0.50 − 1.50 allows for the system to converge 0.49% closer to the ground energy, and 0.37% closer with weights 0.75 − 1.25. Increasing range of weights allows for larger steps in parameter updates, which can cause problems of converging on local-minima due to step size being too large. Continuing the discussion on machine-bias mitigation, the introduction of weighting further improves the ability of our system to dampen quantum machine noise at real-time. Given that we calculate our weight based on noise statistics, any time dependent drift will lower the impact of the quantum machine. Any noisier machine will contribute less, until it is calibrated back to a better noise standard. This benefit is very similar to that of boosting in classical machine learning, where better results are amplified and gradually dominate.

5.5 QAOA Evaluation

In this section, we demonstrate the Quantum Approximate Optimization Algorithm (QAOA) using a MaxCut problem. QAOA is a quantum algorithm that is used for approximating solutions for combinatorial optimization problems, e.g. MaxCut. In our example, we demonstrate the MaxCut problem over the same graph structure discussed prior with $V = [1, 2, 3, 4]$ and $E = [(1, 2), (2, 3), (3, 4), (1, 4)]$. Given a graph G(V,E), where $|V| = n$ and with undirected edge weights of $w_{i,j} > 0$, the MaxCut problem optimizes towards the partitioning nodes into two sets, such that the number of edges E connecting two nodes V_i and V_j from different subsets is maximized. This is formalized in the characteristic Equation 5.

$$C(x) = \sum_{i,j} w_{i,j} x_i (1 - x_j) \quad (5)$$

Where, in Equation 5, x_i is the group of node i, and x_j is the group of node j. Translating this optimization problem to a diagonal Hamiltonian such that it can be optimized via variational quantum algorithms, it can be mapped as a sum over the edge set E.

$$C(x) = \sum_{(i,j)\subset E} (x_i(1 - x_j) + x_j(1 - x_i)) \quad (6)$$

Equation 6 is mapped to a spin Hamiltonian, where $x_i = \frac{1-Z_i}{2}$, as well as swapped from a maximization to a minimization problem, resulting in Equation 7.

$$H = -\sum_{j,k)\subset E} \frac{1}{2}(1 - Z_j Z_k) \quad (7)$$

We employ this optimization problem over the graph discussed prior on the IBMQ machines `Toronto`, `Santiago`, `Quito`, `Lima`, `Casablanca`, `Bogota`, `Manila` and `Belem` both independently and utilizing the EQC framework. The circuit trained is illustrated in Figure 10, which is motivated by the design outlined in the original QAOA paper [16]. In this circuit, a total of 2 parameters are to be optimized, whilst making use of 8 asynchronous systems. In this experiment, we demonstrate the ability of our system to converge with reduced ground state errors at a substantially higher training performance over single machines, highlighting our noise dampening and system throughput contribution.

We illustrate the optimization process in Figure 11. EQC converges under similar iterations when compared with the independent quantum processors. Due to the long experiment run times, there is variation in day to day performance, which can lead to difficulty in directly comparing systems. As a concrete example,

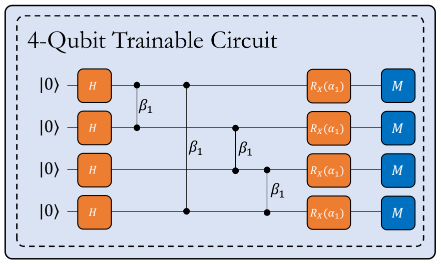

Figure 10: 4-Qubit QAOA circuit. Connected dots represent ZZ gates parameterized by β. R_X represent rotations around the X-axis by parameter α. H represents Hadamard gates. M gates represent measurements.

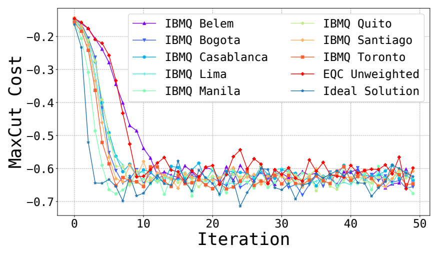

Figure 11: 4-Node unweighted graph MaxCut optimization. Shaded regions represent error over 3 experiment runs. Graph illustrates comparison between IBMQ independent machines and an unweighted EQC system

`Toronto` trains over multiple days, with multiple calibration cycles, whereas `Belem` can can complete training in one hour. EQC performs at a speed of 135,510.2% faster than the slowest IBMQ machine and 322.4% faster than the fastest machine. With respect to converged performance with system weighting, we compare the best solutions presented by our models in Figure 12. As can be seen in Figure 12, by applying a weighting system based on system quality, the model converges quicker than without a weighting system, and to a lower final MaxCut cost. Without weighting, EQC attains convergence second to worst, only beating out IBMQ `Casablanca`. However, by applying our weighting schema, EQC is able to attain solutions close to that of the top 3 IBMQ machines and outperform many worse machines. Weighting approaches improved solutions by 2.863% for a 0.5-1.5 weighting, and 2.343% for 0.25-1.75. In comparison, using the most performant machine of IBMQ `Quito`, we observed a 4.786% improvement over unweighted EQC. This evidence acts as a strong support for EQC in bolstering distributed quantum computing performance with QPU-calibration based system management.

6 DISCUSSION

EQC has proven to provide a substantial speedup in training variational quantum routines whilst providing a framework for real time noise dampening. By viewing multiple QPUs as an ensemble, we are able to weight QPUs with lower confidence, when they are trapped in a noisy condition, and increase the confidence when they are recovered or calibrated. A good example is the training of VQE on

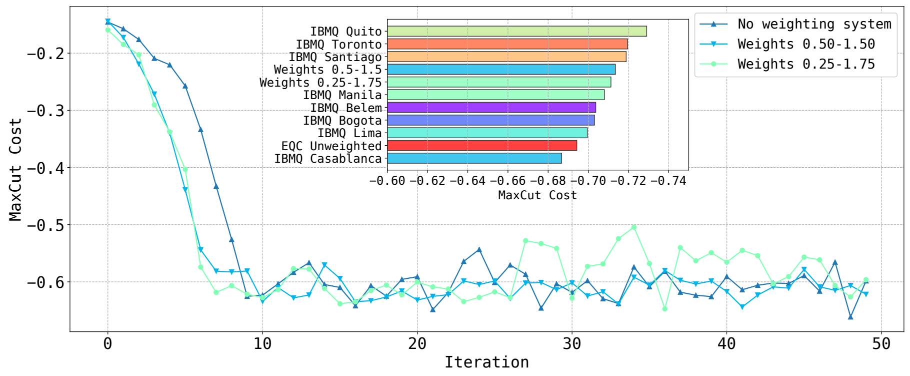

Figure 12: 4-Node unweighted graph MaxCut optimization. Minimum MaxCut cost represents the lowest MaxCut cost attained. Graph illustrates comparison between unweighted and weighted EQC.

IBMQ `Casablanca`. `Casablanca` was able to train faster than any other QPUs on IBMQ, as shown in Figure 6. However, it diverges after being converged for about 20 epochs due to the increasingly deleterious running condition of the machine. For single machine execution, such an impact is inevitable. However, in an ensemble situation, the noise would be recognized and dampened, thereby mitigating this problem. This fluctuating condition of QPU performance highlights the necessity of our ensemble-method for today's erroneous and noisy NISQ devices. Additionally, with our weighting system further applied, it is possible to ensemble many noisy quantum machines alongside a few well-behaved QPUs to create a virtualized quantum device with high-fidelity and performance.

7 RELATED WORK

Understanding quantum device running behavior and mitigating the errors through algorithms, compilation and architectural approaches is undoubtedly necessary for the success of QC in the NISQ era.

Regarding the characterization of NISQ devices, Patel et al. [35] profile the error and execution time of the IBMQ NISQ devices through benchmarkings and draw nine interesting observations. Cross et al. [12] define "quantum volume", which quantifies the largest random circuit of equal width and depth that a NISQ device can successfully implement, concerning both gate and SPAM errors. Murali et al [31] characterize the crosstalk noise of IBMQ devices through random benchmarking, and propose compiler techniques to mitigate the crosstalk impact. Sun and Geller [48] characterize the SPAM error of IBMQ and Rigetti platforms. McEwen et. al. [28] demonstrate NISQ devices susceptibility to cosmic ray error bursts.

Regarding the error mitigation techniques, Tannu et al. [51] observes that the same qubit mapping is reused for all circuit execution trials, leading to correlated errors to be ever prevalent with a certain errors being amplified to give the incorrect results. Motivated by the concept of diversity, they propose an ensemble of *diverse mappings* (EDM) for different trials of QC execution to avoid the dominance of single error type, providing local single machine error mitigation techniques. Our work is analogous in that we also attempt to alleviate NISQ error through mixture. However, we focus on device-specific bias, and leverage multi-devices simultaneously for VQA training, significantly speedup the execution. Patel and Tiwari [36] focus on estimating correct output from raw erroneous outputs of the NISQ devices through statistical methods. Comparatively, we propose a weighting system to regularize the gradient from the multiple backends based on their device properties and runtime conditions.

Regarding the acceleration and scaling of NISQ-based QC, Ravi et al. [38] mentions the utilisation of a vendor-employed machine-aware management system to assist in balancing provider wide load balancing. Li et al. [25] focus on VQE, and present an application-compiler-hardware codesign for QC program compression with little loss of accuracy. Additionally, they tailor the compiler and architecture according to the problem's Pauli strings, greatly reducing the execution overhead. Patel et al. [36] proposes VERITAS, a distributed quantum computing framework for estimating correct outputs of quantum routines, which can provide substantially useful to single-output-distribution algorithms or tasks on quantum computers. Tannu et. al. [51] proposes circuit concurrency for VQA training, however lacks real world evaluation and relies on noise models for benchmarking. [39] Resch et. al. simulates multiple quantum processors working in concurrency, however does not address the uneven noise profile of NISQ-era quantum processors. Tang et al. [50] introduce *CutQC*, which can decompose a large quantum circuit into several pieces which can then be evaluated on small-scale NISQ devices. The probability distribution of the subcircuits are collected for reconstructing the original distribution. Gokhale et al. [19] aim at accelerating pulse generation for VQA. They propose to pregenerate the optimal pulses for certain blocks of gate tailored towards VQA circuits, serving as partial compilation for accelerating the classical GRAPE [18] based machine control pulse generation. Britt and Humble [5] investigate the potential architectural design choices, such as CPU/QPU interconnection, for QPU-integrated HPC. Finally, Das et al. [14] propose to multi-program the same NISQ devices for enhancing NISQ device utilization. In order to mitigate the possible interference, three techniques are presented: picking up the reliable regions; unifying the circuit length; monitoring the reliability and disable multi-programming if necessary. All these techniques, however, are perpendicular to EQC and can be integrated to further mitigate NISQ error, promote utilization rate, and accelerate VQA training speed. For example, if an advanced device (e.g. `IBMQ Toronto`) can sustain more than one VQA circuit simultaneously, multiple jobs can be distributed to the same back-end device for co-execution, further improving the training speed and system utilization.

8 CONCLUSIONS

The field of adaptive ensembled quantum computing has not been tackled in literature to the best of our knowledge. We present the first framework for adaptive quantum ensemble for more accurate and faster VQA training. Our system is by no means the perfect solution, but serves as a promising start to the idea of ensembled quantum computing (EQC) where various homogeneous or heterogeneous quantum devices can cooperate on a single quantum computing application, harvesting accuracy and performance.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Co-design Center for Quantum Advantage (C^2QA) under contract number DESC0012704. Bo Peng and Karol Kowalski were supported by the "Embedding QC into Many-body Frameworks for Strongly Correlated Molecular and Materials Systems" project, which is funded by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences (BES), the Division of Chemical Sciences, Geosciences, and Biosciences. Yufei Ding was supported by NSF 2048144.

APPENDIX: CONVERGENCE PROOF

To show that EQC converges, we make the following assumptions: **(I):** The loss function ℓ must be a differentiable function with $\ell : \mathbb{R}^d \mapsto \mathbb{R}$, based on the expectation values of observables $\langle O_i \rangle$ from the quantum routine; **(II):** The application circuit must be in some part comprised of variational gates with some measurements performed, which means

$$\ell([\vec{\theta}]) = \ell(\langle \psi(\vec{\theta})| \, O_1 \, |\psi(\vec{\theta})\rangle, \ldots, \langle \psi(\vec{\theta})| \, O_m \, |\psi(\vec{\theta})\rangle)$$

for quantum states $|\psi(\vec{\theta})\rangle$ that are produced using the variational circuit for parameters $\vec{\theta} \in \mathbb{R}^d$ is from a fiducial input state $|0\rangle$; **(III):** The loss function should be easily computable on a classical computer Under these assumptions, we show that our asynchronous EQC system can converge. This proof is motivated by existing works [32, 49]. If the system is parameterized by $[\vec{\theta}], \theta \subseteq R^d$, and some cost function $\ell : R^d \to R$, and optimized through stochastic gradient descent rule, we update parameters by $\theta_i^{t+1} = \theta_i^t - \alpha g^t(\theta_i^t)$, where g^t is an unbiased gradient estimator for the variable θ_i, and hence $\mathbf{E}(g^t(\theta_i^t)) = \nabla \ell(\theta_i^t)$. Let's assume ℓ is based on an observable taking the form of: $\ell([\vec{\theta}], \langle O_1 \rangle_{[\vec{\theta}]}, \langle O_2 \rangle_{[\vec{\theta}]}, ..., \langle O_i \rangle_{[\vec{\theta}]})$ where $\langle O_i \rangle_{[\vec{\theta}]}$ is the expected value of O_i generated from the circuit parameterized by $[\vec{\theta}]$. Without losing generality, we assume that for all parameters in $[\vec{\theta}]$, the unbiased estimator of the partial derivative with respect to θ_i is calculated using the *parameter shift rule* $\frac{\delta \ell}{\delta \theta_i} = r[\ell(\theta_i + \frac{\pi}{4r} - \ell(\theta_i - \frac{\pi}{4r})]$ where for common operators such as R_X, R_Y, R_Z, $r = 1/2$. We assume that the system is measuring in a basis comprised of two orthonormal eigenvectors as required by the parameter shift rule, e.g., measuring against $|0\rangle$ and $|1\rangle$.

Since evaluating the exact expectation values from quantum applications becomes rapidly infeasible, we use the sampling mean estimator $\langle o_i \rangle$ of the output $\langle O_i \rangle$. We assume sufficient samples are taken such that $\langle o_i \rangle = \langle O_i \rangle$, and can be used as an unbiased estimator. Given that we will operate on each parameter ϵ times, and that the probability that parameter θ_i fails to be differentiated at any point is p , the probability for a sequence of successes and failures can be expressed as:

$$P_{Setting} = (1-p)^{\epsilon - n} p^n \tag{8}$$

Based on this, the probability of all possible combinations of failures and successes with a set number of failures is:

$$P_{n-failures} = \frac{\epsilon!}{(\epsilon - n)!n!}(1-p)^{\epsilon - n} p^n \tag{9}$$

The probability of the parameter failing n times or less is:

$$P = \sum_{j=0}^{n} \frac{\epsilon!}{(\epsilon - j)!j!}(1-p)^{\epsilon - j}(p^{\epsilon}) \tag{10}$$

If p is reasonably small, and ϵ is sufficiently large as in our case, we can see that with $p \to 0, \epsilon \to \infty, P \to 0$.

We now discuss the asynchronous SGD. In our case, given the significant difference in QPU execution time, we need to rely on the asynchronous gradient descent update rule to $\theta_i^{t+1} = \theta_i^t - \alpha g^{\tau}(\theta_i^{\tau})$, where $\tau \leq t$ and $t - \tau$ is the time delay of the gradient. We assume the stepsize α is non-increasing, and the time delay $t - \tau$ is bounded by some unknown positive integer D such that the model is only a partially asynchronous method [3]. Consequently, the EQC optimization process follows equal distribution of parameter optimization, as all of the parameters are optimized an equal number of times by cycling over each parameter sequentially, where the system operates on each parameter $1, 2, ..., m$ and repeats starting from 1 again after operating on m. Furthermore, given a sequence obtained by sampling from the cyclic sequence, we denote the sequence used in calculating the updated parameters as $\pi(t)$. Therefore, for some positive integer T, we maintain the validity of $|\pi(t) - t| \leq T$ for $\forall t = 1, 2, ...$ (i.e. the asynchronous nature of the system.)

Let's assume the the gradients are bound by a limit of C which means $||g|| \leq C$, $\forall g \subseteq \frac{\delta \ell}{\delta \theta} \forall \theta$. Therefore, the expectations of quantum systems are strictly bound by $0 \leq \langle O \rangle \leq 1$ and requiring $\ell(\langle O \rangle)$ to be bound, hence $||g||$ is bounded. We further argue that the difference between updated parameters calculated from time τ on parameters at time t is bounded by D such that $|t - \tau| \leq D$, $\forall t = 0, 1, 2, ...$, etc. Then with respect to [32], it can be shown that, if $\ell^* = -\infty$,

$$\lim_{t \to \infty} \ell([\vec{\theta}]) = -\infty \tag{11}$$

otherwise, if ℓ^* is finite:

$$\lim_{t \to \infty} \ell([\vec{\theta}]) = \ell^* + mC^2(\frac{1}{2} + m + 2D + T)\alpha \tag{12}$$

where ℓ^* is the optimal solution to the objective function, which demonstrates the convergence.

REFERENCES

[1] 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.

[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Sergio Boixo, Michael Broughton, Bob B Buckley, David A Buell, et al. 2020. Hartree-Fock on a superconducting qubit quantum computer. *Science* 369, 6507 (2020), 1084–1089.

[3] Dimitri P Bertsekas and John N Tsitsiklis. 1989. *Parallel and distributed computation: numerical methods.* Prentice-Hall, Inc.

[4] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549, 7671 (2017), 195–202.

[5] Keith A Britt and Travis S Humble. 2017. High-performance computing with quantum processing units. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13, 3 (2017), 1–13.

[6] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[7] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. 2019. Quantum chemistry in the age of quantum computing. *Chemical reviews* 119, 19 (2019), 10856–10915.

[8] Davide Castelvecchi. 2017. IBM's quantum cloud computer goes commercial. *Nature News* 543, 7644 (2017), 159.

[9] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. 2021. Variational quantum algorithms. *Nature Reviews Physics* 3, 9 (2021), 625–644.

[10] M Cerezo, Kunal Sharma, Andrew Arrasmith, and Patrick J Coles. 2020. Variational quantum state eigensolver. *arXiv preprint arXiv:2004.01372* (2020).

[11] Gavin E Crooks. 2018. Performance of the quantum approximate optimization algorithm on the maximum cut problem. *arXiv preprint arXiv:1811.08419* (2018).

[12] Andrew W Cross, Lev S Bishop, Sarah Sheldon, Paul D Nation, and Jay M Gambetta. 2019. Validating quantum computers using randomized model circuits. *Physical Review A* 100, 3 (2019), 032328.

[13] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. 2017. Open quantum assembly language. *arXiv preprint arXiv:1707.03429* (2017).

[14] 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.* 291–303.

[15] Suguru Endo, Zhenyu Cai, Simon C Benjamin, and Xiao Yuan. 2021. Hybrid quantum-classical algorithms and quantum error mitigation. *Journal of the Physical Society of Japan* 90, 3 (2021), 032001.

[16] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).

[17] Konstantinos Georgopoulos, Clive Emary, and Paolo Zuliani. 2021. Modelling and Simulating the Noisy Behaviour of Near-term Quantum Computers. *arXiv preprint arXiv:2101.02109* (2021).

[18] Steffen J Glaser, Ugo Boscain, Tommaso Calarco, Christiane P Koch, Walter Köckenberger, Ronnie Kosloff, Ilya Kuprov, Burkhard Luy, Sophie Schirmer, Thomas Schulte-Herbrüggen, et al. 2015. Training Schrödinger's cat: quantum optimal control. *The European Physical Journal D* 69, 12 (2015), 1–24.

[19] 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.* 266–278.

[20] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing.* 212–219.

[21] Kathleen E Hamilton, Tyler Kharazi, Titus Morris, Alexander J McCaskey, Ryan S Bennink, and Raphael C Pooser. 2020. Scalable quantum processor noise characterization. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE).* IEEE, 430–440.

[22] Weiwen Jiang, Jinjun Xiong, and Yiyu Shi. 2021. A co-design framework of neural networks and quantum circuits towards quantum advantage. *Nature communications* 12, 1 (2021), 1–13.

[23] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. 2017. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* 549, 7671 (2017), 242–246.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.

[25] Gushu Li, Yunong Shi, and Ali Javadi-Abhari. 2021. Software-Hardware Co-Optimization for Computational Chemistry on Superconducting Quantum Processors. *arXiv preprint arXiv:2105.07127* (2021).

[26] Hsuan-Hao Lu, Natalie Klco, Joseph M Lukens, Titus D Morris, Aaina Bansal, Andreas Ekström, Gaute Hagen, Thomas Papenbrock, Andrew M Weiner, Martin J Savage, et al. 2019. Simulations of subatomic many-body physics on a quantum frequency processor. *Physical Review A* 100, 1 (2019), 012320.

[27] Pushpa Mannava. 2014. An Overview of Cloud Computing and Deployment of Big Data Analytics in the Cloud. *International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Online ISSN* (2014), 2394–4099.

[28] Matt McEwen, Lara Faoro, Kunal Arya, Andrew Dunsworth, Trent Huang, Seon Kim, Brian Burkett, Austin Fowler, Frank Arute, Joseph C Bardin, et al. 2021. Resolving catastrophic error bursts from cosmic rays in large arrays of superconducting qubits. *Nature Physics* (2021), 1–5.

[29] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18).* 561–577.

[30] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. 2019. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA).* IEEE, 527–540.

[31] Prakash Murali, David C McKay, Margaret Martonosi, and Ali Javadi-Abhari. 2020. Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems.* 1001–1016.

[32] Angelia Nedić, Dimitri P Bertsekas, and Vivek S Borkar. 2001. Distributed asynchronous incremental subgradient methods. *Studies in Computational Mathematics* 8, C (2001), 381–407.

[33] John L Orrell and Ben Loer. 2020. Sensor-assisted fault mitigation in quantum computation. *arXiv preprint arXiv:2012.12423* (2020).

[34] Robert M Parrish, Edward G Hohenstein, Peter L McMahon, and Todd J Martínez. 2019. Quantum computation of electronic transitions using a variational quantum eigensolver. *Physical review letters* 122, 23 (2019), 230401.

[35] Tirthak Patel, Abhay Potharaju, Baolin Li, Rohan Basu Roy, and Devesh Tiwari. 2020. Experimental evaluation of NISQ quantum computers: error measurement, characterization, and implications. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE, 1–15.

[36] Tirthak Patel and Devesh Tiwari. 2020. Veritas: accurately estimating the correct output on noisy intermediate-scale quantum computers. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE, 1–16.

[37] A Peruzzo et al. 2013. A variational eigenvalue solver on a quantum processor. eprint. *arXiv preprint arXiv:1304.3061* (2013).

[38] Gokul Subramanian Ravi, Kaitlin N Smith, Pranav Gokhale, and Frederic T Chong. 2021. Quantum Computing in the Cloud: Analyzing job and machine characteristics. In *2021 IEEE International Symposium on Workload Characterization (IISWC).* IEEE, 39–50.

[39] Salonik Resch, Anthony Gutierrez, Joon Suk Huh, Srikant Bharadwaj, Yasuko Eckert, Gabriel Loh, Mark Oskin, and Swamit Tannu. 2021. Accelerating Variational Quantum Algorithms Using Circuit Concurrency. *arXiv preprint arXiv:2109.01714* (2021).

[40] Alessandro Roggero, Andy CY Li, Joseph Carlson, Rajan Gupta, and Gabriel N Perdue. 2020. Quantum computing for neutrino-nucleus scattering. *Physical Review D* 101, 7 (2020), 074038.

[41] Maria Schuld and Nathan Killoran. 2019. Quantum machine learning in feature hilbert spaces. *Physical review letters* 122, 4 (2019), 040504.

[42] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. 2014. The quest for a quantum neural network. *Quantum Information Processing* 13, 11 (2014), 2567–2586.

[43] Sarah Sheldon, Easwar Magesan, Jerry M Chow, and Jay M Gambetta. 2016. Procedure for systematically tuning up cross-talk in the cross-resonance gate. *Physical Review A* 93, 6 (2016), 060302.

[44] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.

[45] Samuel A Stein, Betis Baheri, Daniel Chen, Ying Mao, Qiang Guan, Shuai Xu, Caiwen Ding, and Ang Li. 2022. Quclassi: A hybrid deep neural network architecture based on quantum state fidelity. *Proceedings of Machine Learning and Systems* (2022).

[46] Samuel A Stein, Betis Baheri, Ray Marie Tischio, Ying Mao, Qiang Guan, Ang Li, Bo Fang, and Shuai Xu. 2020. Qugan: A generative adversarial network through quantum states. *arXiv preprint arXiv:2010.09036* (2020).

[47] Michael Streif and Martin Leib. 2019. Comparison of QAOA with quantum and simulated annealing. *arXiv preprint arXiv:1901.01903* (2019).

[48] Mingyu Sun and Michael R Geller. 2018. Efficient characterization of correlated SPAM errors. *arXiv preprint arXiv:1810.10523* (2018).

[49] Ryan Sweke, Frederik Wilde, Johannes Meyer, Maria Schuld, Paul K Fährmann, Barthélémy Meynard-Piganeau, and Jens Eisert. 2020. Stochastic gradient descent for hybrid quantum-classical optimization. *Quantum* 4 (2020), 314.

[50] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi. 2021. CutQC: using small quantum computers for large quantum circuit evaluations. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 473–486.

[51] 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*. 253–265.

[52] Caroline Tornow, Naoki Kanazawa, William E. Shanks, and Daniel J. Egger. 2022. Minimum quantum run-time characterization and calibration via restless measurements with dynamic repetition rates. arXiv:arXiv:2202.06981

[53] Andrew Tranter, Peter J Love, Florian Mintert, and Peter V Coveney. 2018. A comparison of the bravyi–kitaev and jordan–wigner transformations for the quantum simulation of quantum chemistry. *Journal of chemical theory and computation* 14, 11 (2018), 5617–5630.

[54] Xiao Yuan, Suguru Endo, Qi Zhao, Ying Li, and Simon C Benjamin. 2019. Theory of variational quantum simulation. *Quantum* 3 (2019), 191.

[55] Shanshan Zhang, Ce Zhang, Zhao You, Rong Zheng, and Bo Xu. 2013. Asynchronous stochastic gradient descent for DNN training. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 6660–6663.