CPEN 400Q Report

1. Introduction

Our paper presents a novel approach called virtual quantum error detection (VQED), which enables the extraction of computational outcomes from a quantum circuit based on error detection results. Although VQED implies the detection of quantum errors, it is primarily used as a quantum error mitigation technique to reduce noise in quantum circuits. The paper begins with a brief overview of stabilizer formalism, followed by an introduction to syndrome measurement, a widely used technique for quantum error detection. Given the considerable overhead associated with syndrome measurement due to the computation required on each stabilizer, we subsequently introduce symmetry expansion, an effective quantum error mitigation strategy. The significance of VQED lies in its ability to generalize the symmetry expansion method, allowing for efficient error detection without the need for executing numerous syndrome measurements.

2. Theory

2.1 Background

Stabilizer formalism is a common method that utilizes the properties of Pauli groups to create circuits for error detection. We can define a n-qubit Pauli group $\mathbf{P_n}$ as follows:

> Let X, Y, and Z be the Pauli Operators and I be the identity matrix.
>
> $$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$
>
> $\mathbf{P_n}$ would be $\{\pm i, \pm 1\} \times \{I, X, Y, Z\}^{\otimes n}$, where $\{\pm i, \pm 1\}$ would be the eigenvalues.

As the main motivation behind quantum error detection is to protect logical qubits from noise, we can create redundancy by encoding a single logical qubit into several physical qubits. To encode k logical qubits into n physical qubits, we need to define stabilizer groups and generator sets. We can define a stabilizer group as:

> $\mathbf{S} = \{S_1, \ldots, S_{2^{n-k}}\} \subset \mathbf{P_n}$, where $\mathbf{S}$ is a commutative subgroup of the Pauli group.

Conversely, we can also define a generator set of that particular stabilizer group as follows:

> $\mathbf{G} = \{G_1, \ldots, G_{n-k}\}$, whereby taking the tensor product of every possible subset of $\mathbf{G}$ is equivalent to the set $\mathbf{S}$.

After defining the stabilizer group and generator set, we can define the logical space of our stabilizer code C as an eigenspace that contains all eigenvectors with an eigenvalue of $+1$ that is common to every element of our stabilizer group S. We can also interpret this as a codespace where applying any one of our stabilizers to this particular state has no effect i.e. $C = \{|\psi\rangle \mid \forall S_i \in S,\ S_i |\psi\rangle = |\psi\rangle\}$. The stabilizer code space is crucial because we can envision it as a code space that is noise-free. When noise is introduced

to our state by an external factor, stabilizers within the quantum circuit project it back to our code space to mitigate the effect of the noise experienced.

As we will also be dealing with several state matrices, we will introduce them here:

ρ_{id} is the ideal density matrix in noiseless conditions.
ρ is ρ_{id} with noise applied.
ρ_{det} is ρ projected to the code space C.

To project a density matrix to our code space C, we can create a projector P by averaging the summation of the stabilizers:

$$P = \frac{1}{2^{n-k}} \sum_{S_i \in S} S_i. \tag{2.1}$$

This projector P is used to project the noisy matrix back onto the code space with the following equation. The resulting ρ_{det} is the representation of ρ with the noise removed in our code space C.

$$\rho_{\text{det}} = \frac{P\rho P}{\text{tr}[\rho P]} \tag{2.2}$$

2.2 Syndrome Measurement

Syndrome measurement is a common quantum error detection technique used to identify and diagnose errors in quantum circuits without measuring the quantum state. This is done by arranging a circuit as follows:

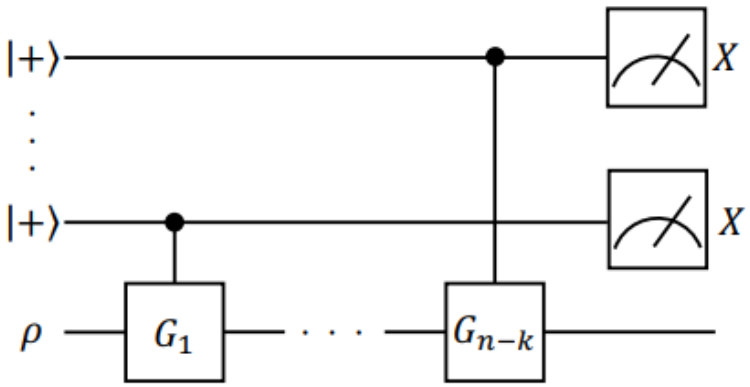

Figure 1: Circuit for Syndrome Measurement[2]

First, we apply a Hadamard gate to first $n-k$ control wires to arrive at a plus state. Then, we input the state that we want to check for errors, represented as a density matrix, and apply the generators based on the value of the control wires. Finally, we measure the control wires and look for any wires that yields a measurement result of -1. This indicates that an error was introduced in the circuit. On the other hand, if all measurement results are +1, we can conclude with a high degree of confidence that there was no error introduced.

2.3 Symmetry Expansion

Symmetry expansion is a quantum error mitigation technique that leverages symmetry in quantum systems to reduce noise or errors without relying on syndrome measurements. The key difference between quantum error correction and quantum error detection is that the former aims to recover the ideal state after the noise has been introduced, while the latter aims to recover the ideal measurement statistics for an observable instead. The expectation value of an observable O on some state ρ_{det} is calculated by $\text{Tr}(\rho_{\text{det}}O)$ (by Born Rule). In the paper they expand this formula to obtain:

$$\text{Tr}(\rho_{\text{det}}O) = \frac{\sum_{S_i \in S} \text{Tr}(\rho O S_i)}{\sum_{S_i \in S} \text{Tr}(\rho S_i)} \tag{2.3}$$

Although this approach may appear complicated at first glance, it is actually quite simple to implement. The steps involved are as follows: randomly select a stabilizer from the set $\mathbf{S}$, measure the noisy state ρ using both S_i and OS_i, and record the measurement outcomes. This process is then repeated N times, and the average outcome is taken across all iterations.

2.4 Virtual Quantum Error Detection

Since symmetry expansion can only be utilized in the state immediately prior to measurement, VQED presents an alternative approach for computing the same error-mitigated expectation values during circuit execution. Rather than selecting any arbitrary state, we utilize the initial ideal density matrix as the zero state with L unitary gates applied to it. We also introduce noise to simulate the effects of noise on our logic gates in this manner:

$$\rho'_{\text{det}} = P \circ \epsilon_L \circ U_L \circ \cdots \circ P \circ \epsilon_1 \circ U_1(\rho_0), where \tag{2.4}$$

ρ_0 is the zero state represented as a density matrix, U is an arbitrary unitary gate, ϵ is some noise in the circuit, and P is the projector to our codespace C

We can then divide by the trace of ρ_{det} to normalize the density matrix and make the probabilities sum to 1:

$$\rho_{\text{det}} = \frac{\rho'_{\text{det}}}{\text{tr}(\rho'_{\text{det}})} \tag{2.5}$$

We can now perform virtual quantum error detection by implementing the following circuit:

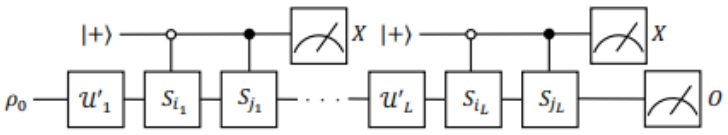

Figure 2: Circuit for Virtual Quantum Error Detection[2]

Similarly to symmetry expansion, we can perform VQED on our noisy circuit by the following steps: randomly select two stabilizers from the set $\mathbf{S}$, run the circuit above,

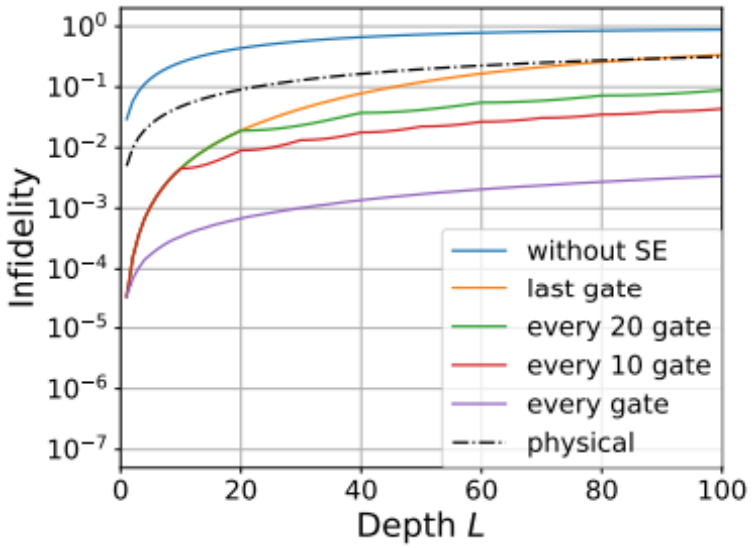

Figure 3: Infidelity vs Depth[2]

record the tensor product of the X measurements as a and the tensor product of a and the O measurement as b. This process is then repeated N times, and the value $\frac{b}{a}$ would be our average expected value.

2.5 Numerical Simulation

After the circuit introduced in VQED is implemented, we can run numerical simulation by selecting arbitrary values for our unitary gates and noise matrices. Furthermore, the authors were able to produce graphs for three different combinations of logical and physical qubits, however, we primarily focused on the example where we try to encode one logical qubit into four physical qubits.
Referencing Figure 3, infidelity is measured by:

$$1 - \mathrm{tr}(\rho_{\mathrm{det}}|\bar{\Psi}\rangle\langle\bar{\Psi}|), \text{ where } \bar{\Psi} \text{ represents the state of the noiseless circuit}$$

Our goal is to measure the degree of deviation between the ideal state and the noisy circuit, which is reflected in the infidelity score. Ideally, we want a low score for infidelity. From the graph, we observe that as the number of unitary gates in the original density matrix increases, the infidelity scores gradually increase. This is because more noise gets introduced during the circuit execution, which deviates our density matrix from the ideal one. Additionally, we can see that more frequent performance of VQED results in lower infidelity scores. This aligns with our intuition that frequent VQED during the circuit execution can help mitigate more errors.

3. Results

3.1 Framework

For our project's framework, we considered using Pennylane, Tensorflow Quantum, and QisKit. After comparing the 3 library's support of mixed states and noise simulation, we found that each framework supported the implementation of the circuits for syndrome

measurement, symmetry expansion, and VQED. Additionally, we did not find any significant performance benefits in any framework. Therefore, we chose to utilize the Pennylane library as it was a tool that our group was familiar with from our current coursework.

Toward the end of our paper, we discovered the author's used the quantum framework QuTiP. Upon initial inspection, we did not find any significant benefits of using QuTiP over Pennylane. However, we later realized that QuTiP was more efficient at measuring the performance of our VQED circuit. This is due to Pennylane's "default.mixed" option having a maximum capacity of 23 wires which prevented us from completing the final numerical simulation of our paper. Given the chance to complete this project again, we would consider switching over to QuTiP.

3.2 Background

For the rest of our measurements, we decided to encode $k = 1$ logical qubit using $n = 3$ physical qubits. Furthermore, we decided to use the generators $G_1 = Z_1Z_2$ and $G_2 = Z_2Z_3$. Consequently, this meant $\mathbf{S} = \{I, G_1, G_2, G_1 \otimes G_2\}$.

3.3 Syndrome Measurement

After choosing our generators, we were able to code up a circuit for syndrome measurement. This function takes in the density matrix that we wish to detect if noise was introduced, the associated generators, as well as the estimation and target wires.

```
dev = qml.device("default.mixed", wires=5, shots=1)
@qml.qnode(dev)
def syndrome_measurement(density_matrix, generators,
                         estimation_wires, target_wires):

    # Prepare target wires as our density matrix
    qml.QubitDensityMatrix(density_matrix, wires=target_wires)

    for wire in estimation_wires:
        qml.Hadamard(wires=wire)

    # Introduce Bit Error on a wire
    qml.BitFlip(1, target_wires[2])

    L = len(estimation_wires)
    for i, generator in enumerate(generators):
        qml.ControlledQubitUnitary(generator, \
            control_wires=estimation_wires[L-1-i], wires=target_wires)

    measurements = []
    for wire in estimation_wires:
        measurements.append(qml.sample(qml.PauliX(wire)))

    return measurements
```

From the generators we chose above, we can decipher which wire a bit flip occurred by referencing the following table:

Z_1Z_2	Z_2Z_3	Error type	Action
+1	+1	no error	no action
+1	−1	bit 3 flipped	flip bit 3
−1	+1	bit 1 flipped	flip bit 1
−1	−1	bit 2 flipped	flip bit 2

Figure 4: Table to detect bit flip errors[1]

We were able to test every combination of bit-flip errors and achieve the correct error type 100% of the time, so we are confident that our implementation of syndrome measurement is correct.

3.4 Symmetry Expansion

To verify we correctly coded the circuit for symmetry expansion, we chose a random density matrix ρ, applied noise to it, and projected it to our codespace C using our projector P, to obtain ρ_{det}, and calculated $\text{Tr}(\rho_{\text{det}}O)$.

```
G1 = qml.Identity(0) @ qml.PauliZ(1) @ qml.PauliZ(2)
G2 = qml.PauliZ(0) @ qml.PauliZ(1) @ qml.Identity(2)

G1 = G1.matrix()
G2 = G2.matrix()

# S size of 4 (Assuming generators that we chose should be able to make
    up the set of S)
I = qml.Identity(0) @ qml.Identity(1) @ qml.Identity(2)
I = I.matrix()

G = [G1, G2]
S = [G1, G2, G1 @ G2, I]

def calculate_P_with_S(S):
    P = np.zeros(S[0].shape)
    for elem in S:
        P += elem
    P *= 1/(len(S))
    return P

P = calculate_P_with_S(S)

def generate_density_matrix():
    state = np.array([0.5, 0.5, 0, 0.5, 0, 0, 0.5,0])
    rho = np.outer(state, np.conj(state).T)
    return rho

dev = qml.device("default.mixed", wires=3)
@qml.qnode(dev)
```

```
def generate_noise(wires):
    density_matrix = generate_density_matrix()
    qml.QubitDensityMatrix(density_matrix, wires=wires)
    qml.BitFlip(0.75,wires[0])
    qml.BitFlip(0.35,wires[1])
    qml.BitFlip(1,wires[2])
    return qml.density_matrix(wires)

# create rho with noise applied
rho =  generate_noise([0,1,2])
rho_det = (P @ rho @ P)/np.trace(rho @ P)

# Create random observable O
O = qml.Identity(0) @ qml.Identity(1) @ qml.PauliZ(2)
O = O.matrix()

# Expected value of the observable
expected = np.trace(rho_det @ O)
print(expected)
```

Afterwards, we were able to simulate $\frac{\sum_{S_i \in S} \mathrm{Tr}(\rho O S_i)}{\sum_{S_i \in S} \mathrm{Tr}(\rho S_i)}$ with the following code:

```
N = 50000
a = 0
b = 0

for i in range(N):
    Si = random.sample(S,1)[0]
    a_s = np.trace(rho @ Si)
    b_s = np.trace(rho @ O @ Si)
    a += a_s
    b += b_s
actual = b/a
print("Expected: " + str(expected))
print("Actual: " + str(actual))
print("Error: " + str(abs((actual-expected)/expected)*100))
```

Running the code above, we were able to get the following results:

```
Expected: (-0.25925925925925336+0j)
Actual: (-0.2604649917124443+0j)
Error: 0.465068231945089
```

Since the error rate is generally less than 1%, we were confident to say that our implementation of symmetry expansion was correct.

3.5 Virtual Quantum Error Detection

To verify that we implemented the circuit correctly for VQED, we were able to verify the results obtained from Symmetry Expansion and cross-reference it with the results obtained from running the circuit. The VQED code is as follows:

```
L = len(U_list)
rho_naught = generate_density_matrix()
estimation_wires = list(range(0, L))
target_wires = list(range(L, L+3))

dev = qml.device("default.mixed", wires=(len(estimation_wires+
    target_wires)))
```

```
@qml.qnode(dev)
def VQED(stabilizer_list, unitary_list):
    # stabilizer list is same len as estimation wires

    # initialize target and estimation wires
    qml.QubitDensityMatrix(rho_naught, wires=target_wires)
    for wire in estimation_wires:
        qml.Hadamard(wires=wire)

    for l in estimation_wires:
        # get the 3 unitary matries
        s_i = stabilizer_list[l][0]
        s_j = stabilizer_list[l][1]
        U = unitary_list[l]

        # apply the gates
        qml.QubitUnitary(U, wires=target_wires) # Apply U,
        qml.BitFlip(0.5, wires=target_wires[0]) # add some noise (U')
        qml.ControlledQubitUnitary(s_i, control_wires=estimation_wires[l], wires=target_wires, control_values="0")
        qml.ControlledQubitUnitary(s_j, control_wires=estimation_wires[l], wires=target_wires, control_values="1")
    prod = qml.PauliX(estimation_wires[0])
    for wire in range(1,len(estimation_wires)):
        prod = prod @ qml.PauliX(estimation_wires[wire])
    return (qml.expval(prod),
            qml.expval(qml.Hermitian(O, target_wires)))
```

After coding the circuit, we were able to run it N times to get the average outcome over all iterations:

```
a = 0
b = 0
N = 100
for s in range(N):
    s_sample = [random.sample(S,2) for i in range(L)]
    a_s, O_measurement = VQED(s_sample, U_list)
    b_s = a_s * O_measurement
    a += a_s
    b += b_s

a = a/N
b = b/N

print("Actual Expval:")
actual_expval = b / a
print(actual_expval)

# Expected expval is similarly calculated above for symmetry expansion
expected_expval = get_real_expval()
print("Expected Expval:")
print(expected_expval)
percent_error = abs((actual_expval - expected_expval) / expected_expval) * 100
percent_error = percent_error.real
print(f"% error: {percent_error}%")
```

Running the code above, we were able to get the following results:

```
Actual Expval:
-0.5000000000000291
Expected Expval:
(-0.4999999999999999+0j)
% error: 5.839773109528325e-12%
```

We were also able to simulate the circuit with different values for our unitary gates and noise to ensure that our circuit was coded correctly.

3.6 Numerical Analysis

In order to conduct numerical analysis, we needed to run our VQED circuit with varying numbers of unitary gates. However, as discussed in the Framework and Reproducibility section, we were unable to replicate the results presented in the paper due to hardware limitations.

3.7 Reproducibility

In our attempt to implement the methods outlined in our paper, we encountered several challenges. The first significant issue arose from the ambiguous notation employed by the authors in certain equation definitions. Variables were introduced without prior definitions, necessitating us to make assumptions based on our best interpretation. We made efforts to contact the authors for clarification on a suspected typo, but unfortunately, we were unable to establish communication with them.

We also had some reproducibility issues with creating the circuits illustrated in the paper. The instructions for setting up the various quantum circuits were minimal, resulting in our group needing to make multiple attempts to recreate the visual representations.

As students without an extensive background in advanced mathematics, our group encountered difficulties with the level of language used to describe mathematical concepts in our paper. The authors assumed a high level of reader understanding, which resulted in a significant amount of time spent trying to comprehend the theoretical aspects.

$$\begin{aligned} \rho_{\mathrm{bf}} &= \frac{1}{2^L} \sum_{\boldsymbol{pq}} |\boldsymbol{p}\rangle \langle \boldsymbol{q}| \otimes \rho_{\boldsymbol{ij}}^{\boldsymbol{pq}}, \\ \rho_{\boldsymbol{ij}}^{\boldsymbol{pq}} &= \mathcal{P}_{i_L j_L}^{p_L q_L} \circ \mathcal{E}_L \circ \mathcal{U}_L \circ \cdots \circ \mathcal{P}_{i_1 j_1}^{p_1 q_1} \circ \mathcal{E}_1 \circ \mathcal{U}_1(\rho_0), \end{aligned} \tag{9}$$

where $\boldsymbol{p}$ and $\boldsymbol{q}$ are bitstrings of length L and

$$\mathcal{P}_{i_l j_l}^{p_l q_l}(\cdot) = S_{j_l}^{p_l} S_{i_l}^{1-p_l} \cdot S_{i_l}^{1-q_l} S_{j_l}^{q_l}. \tag{10}$$

Then, the expectation value of the observable $X^{\otimes L} \otimes O$ in this state is:

$$\begin{aligned} \langle X^{\otimes L} \otimes O \rangle &= \mathrm{tr}\left[\rho_{\mathrm{bf}} X^{\otimes L} \otimes O\right] \\ &= \frac{1}{2^L} \sum_{\boldsymbol{p}} \mathrm{tr}[\rho_{\boldsymbol{ij}}^{\boldsymbol{pp+1}} O] \end{aligned} \tag{11}$$

Figure 4: Example of Complex Equations[1]

Relating to results, we were able to replicate Virtual Quantum Error Detection with our chosen library, however we were unable to reproduce the graph depicted in Figure 2.3. Specifically, the Pennylane device "default.mixed" only supports up to 23 wires which prevented us from replicating results for larger values of L. Furthermore, we faced additional hardware constraints, as even a depth of 10 required approximately 20 minutes to run, rendering larger values of L impractical. We did explore several other frameworks such as QuTiP, but could not port our Pennylane code over within the given timeframe.

4. Conclusion

In conclusion, we were able to successfully replicate the circuits used for syndrome measurement, symmetry expansion, and virtual quantum error detection. Additionally, we conducted various numerical tests to ensure the accuracy of our circuit implementation. However, due to hardware limitations and the chosen framework, we were unable to replicate the graphs presented in the original paper. We also observed that the VQED method is similar to a quantum error mitigation technique rather than an error detection method, as stated in the paper, which suggests that "VQED can be considered as a QEM method implemented on the code space"[2]. Overall, the VQED method shows promise in mitigating error loss and has certain advantages over symmetry expansion. Despite the limitations faced during our implementation, this research has contributed towards further understanding the potential of VQED.

5. References

[1] Nielsen, M. A. & Chuang, I. L. (n.d.). Quantum algorithms via linear algebra - cas. Retrieved April 13, 2023, from http://mmrc.amss.cas.cn/tlb/201702/W020170224608149911380.pdf

[2] Tsubouchi, K., Suzuki, Y., Tokunaga, Y., Yoshioka, N., & Endo, S. (2023, February 6). *Virtual quantum error detection. arXiv.org.* Retrieved April 13, 2023, from https://arxiv.org/abs/2302.02626