Implementation of the Projected Variational Quantum Dynamics Algorithm

G. Adam Cox, I-Tung Chen, Sanskriti Joshi, Tommy Nguyen

February 28, 2023

1 Introduction

Quantum simulations are a fundamental way to understand quantum phenomena. Researchers try to understand quantum phenomena experimentally by building up systems with qubits and letting their system evolve according to given constraints and variables. However, systems with many degrees of freedom are hard to create experiments for due to the vast space of variables that need to be controlled. This is why simulating quantum systems on computers is a very powerful tool if it can be done correctly and efficiently. Additionally, since quantum systems are inherently "quantum", it is believed that these systems are best simulated on a quantum computer as they have the nature of being quantum themselves and therefore might be easier to simulate by leveraging this quantumness (entanglement, coherence, etc.). In QHACK, we probe this idea by studying quantum simulations, but more specifically, the time dynamics of quantum systems as governed by the Schrodinger equation on quantum computers:

$$\frac{i\partial \left|\Psi\right\rangle}{\partial t} = \hat{H} \left|\Psi\right\rangle \tag{1}$$

We stick to time-independent hamiltonians H as this is the easier case out of the two, albeit still NP-hard :), and this gives the solution:

$$\left|\Psi(t)\right\rangle = e^{-i\hat{H}t} \left|\Psi(0\right\rangle \tag{2}$$

As referenced in this Nature paper two months ago [3], the problem of trying to implement the above equation is very tricky due to H being almost anything you want and is a huge problem for **quantum computing today**! There are two main approaches to tackle this problem: variational and decomposition techniques which each have many subfields within them. We will not give an overview of the different methods as the Nature paper does this already, but we instead focus on a specifc algorithm within the variational umbrella: the projected variational quantum dynamics algorithm (pVQD) as discovered by Stefano Barison a year ago [1]. The novel idea of this algorithm is that it evolves variational parameters by optimizing them at each time step to resemble a small Trotter step using gradient descent. And so this leverages **hybrid classical-quantum computing** techniques for efficient computation. Stefano has shown that pVQD is an improvement over the time-dependent variational algorithm [2] by being global: updating all parameters at once, and has linear scaling with respect to the number of parameters. However, in [1], pVQD has only been used for quantum simulations on a spin chain system and hasn't been tested on actual quantum hardware. In this hackathon, we attempt a couple of novel approaches which has been divided into the following sections below:

1. Reproduce the results found in the figures in [1] with qiskit and test the reproducibility.
2. Code the pVQD algorithm with Pennylane's quantum language which is suited for quantum machine learning applications. Since pVQD has a gradient descent step, we believe that Pennylane's built-in functionality to incorporate machine learning techniques will improve the efficiency of this algorithm for actual runtime on a quantum simulator and qpu.
3. Run the algorithm on an actual quantum hardware device: IBM's 7 qubit Nairobi machine, and compare it to the previous simulations above.

2 Short Background to pVQD

This section has been added here for coherence throughout this paper. A more detailed description of the algorithm should be looked at in [1]. Here we just go over the main ideas.

2.1 pVQD Algorithm

Our goal is to simulate eq.2. We do this by starting with a parameterized ansatz state $\left|\psi_{w(t)}\right\rangle = \prod_d \prod_i R_\sigma^{i,d}(w_i(t)) \left|\psi_0\right\rangle$ where $w(t) \in \mathbb{R}^p$ is a vector of p parameters describing our state at each time t for each ith qubit. R_σ is a rotation around the axis σ and d is the depth of the circuit (how many times its gets repeated). After each trotter step δt, our time evolved state is:

$$\left|\phi(t+\delta t)\right\rangle = e^{-i\hat{H}\delta t}\left|\psi_{w(t)}\right\rangle \tag{3}$$

The pVQD aims to maximize:

$$argmax_{dw} \quad |\left\langle\phi(t+\delta t)\middle|\psi_{w+dw}\right\rangle|^2 \tag{4}$$

by finding a dw that does this. This is done through a gradient descent step:

$$dw^{new} = dw^{old} - \eta\nabla_{dw}L(dw^{old}, \delta t) \tag{5}$$

where L is the cost function we aim to minimize which describes the step in-fidelity:

$$L(dw, \delta t) = \frac{1 - |\left\langle\phi(t+\delta t)\middle|\psi_{w+dw}\right\rangle|^2}{\delta t^2} \tag{6}$$

where η is the typical learning rate in ML.

2.2 Model

The model that we simulate is the transverse ising field model as the one done in the paper:

$$H = J\Sigma_{i=1}^N \sigma_i^z \sigma_{i+1}^z + h\Sigma_{i=1}^N \sigma_i^x \tag{7}$$

where we use N = 3 qubits, J = 1/4, h=1.

3 Implementation with Qiskit [1]

The source code that [1] uses can be found here: https://github.com/StefanoBarison/p-VQD while our adaptation of this code in qiskit can be found here: https://github.com/NnguyenHTommy/p-VQD. We reproduce Fig.2 and Fig.3 found in [1] with our own Fig.1 and Fig.2. Fig.1 is a plot of the total infidelity between our simulated pVQD state and the exact state over all time steps. We don't know how the exact state was computed but we were given the exact state at each time step from the github. We see that as we increase the number of samples (shots), the average infidelity decreases which makes sense because our statistics are better. However, we see that we don't really get an improvement until we transition from 10^6 to 10^7 samples. This is in contrast to Fig. 2 in their paper which we believe is because we only took one measurement at each amount of samples while they took 10 measurements. Additionally, comparing the two figures, our order of magnitude for the infidelity is 10x lower suggesting that our revised method of calculating the infidelity is better.

Next looking at Fig.2, we can see that as we increase the number of shots, we get closer and closer to the exact results for the magnetization. In fact, already at 8000 shots, the pVQD algorithm does surprisingly well. This is in contrast to the TDVA algorithim which needs 80,000 shots to get results that look like they fit to the exact results [2]. Our results are consistent with Fig.3 in [1] so we know that we are implementing things correctly in qiskit, yet we also run with more varying amount of shots than they did to see how bad/good we can get. We see that with 80 shots, pVQD doesn't give accurate results while 80,000

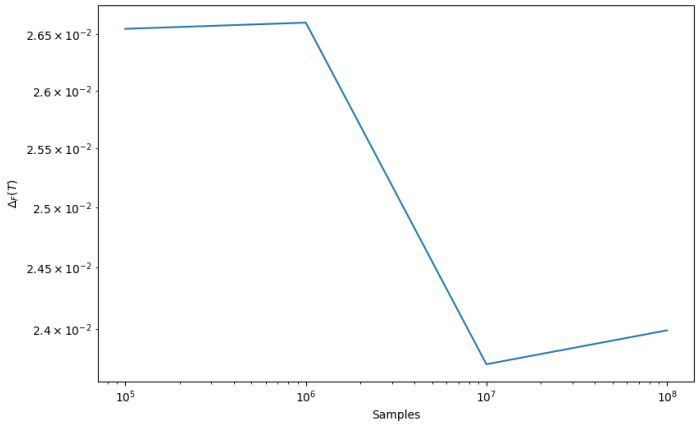

Figure 1: Average infidelity Δ_F over the number of samples over the entire time duration with 60 steps and a depth of 3.

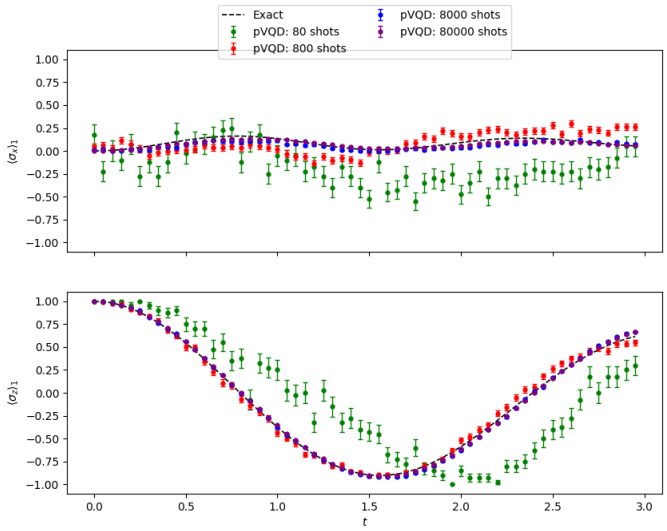

Figure 2: Magnetization measured on qubit 0 with varying amount of shots. (Top) σ_x (Bottom) σ_z

shots is not much better than 8000 shots. This seems to indicate a scaling + saturation law with the number of shots. One thing to also note is that the pVQD algorithm seems to simulate σ_z better than σ_x. We think this is because we apply R_x and R_y rotations for our circuit ansatz and so this would lead to some variance in σ_x but not σ_z. Here we see that 8000 shots are needed for a pretty accurate representation of the exact state and from[1], the pVQD algorithm needs 10x less shots to do this giving more justification that pVQD is better.

4 Implementation with Pennylane

We implemented the pVQD algorithm using Xanadu's open-source quantum machine learning library, 'pennylane' for an open-chain Transverse Field Ising Model consisting of three qubits,

$$H = J\Sigma_{i=0}^{2}\sigma_i^z\sigma_{i+1}^z + h\Sigma_{i=0}^{2}\sigma_i^x \tag{8}$$

for $J = 0.25$ and $h = 1$. We chose this specific system in order to make direct comparison to the previously published work[1].

Implementation of the pVQD algorithm consisted of four components: defining the Hamiltonian, defining our circuit model, defining our cost function, then implementing the optimization loop over our time-span. Our total time span, T, lasted 2.0, with step sizes, δt of 0.05. (Time units are arbitrary.) Our code repository is found here: https://github.com/gadamc/uw_qx_pvqd_qhack2023. Relevant files are 'circuit.py', 'vqd.py', and 'test-vqd.ipynb'.

Our Hamiltonian implementation used 'pennylane''s built-in Hamiltonian function with appropriate coefficients

```
import pennylane as qml

def hamiltonian(coupling_strength=0.25, field_strength=1):
    coefficients = [field_strength]*3
    coefficients += [coupling_strength]*2

    obs = [qml.PauliX(0), qml.PauliX(1), qml.PauliX(2),
           qml.PauliZ(0) @ qml.PauliZ(1),
           qml.PauliZ(1) @ qml.PauliZ(2)]

    return qml.Hamiltonian(coefficients, obs)
```

To apply the Hamiltonian to our state in the algorithm, we utilized the 'pennylane' function 'ApproxTimeEvolution' with a single time step, δt.

Our multilayered circuit, $C(w)$ consisted of a combination of individual qubit rotations about X or Y (alternating at each layer, l) and IsingZZ rotations for the coupled qubits. This is of the form

$$C(w) = \Pi_l \Pi_i R_\alpha^{(i)}(w_{i,l}) \Pi_j e^{-i w_{j,l} \sigma_j^z \sigma_{j+1}^z} \tag{9}$$

where α is either x or y, alternating between layers, l.

This circuit can be found here https://github.com/gadamc/uw_qx_pvqd_qhack2023/blob/main/circuit.py#L15 and is visualized in Fig. 3.

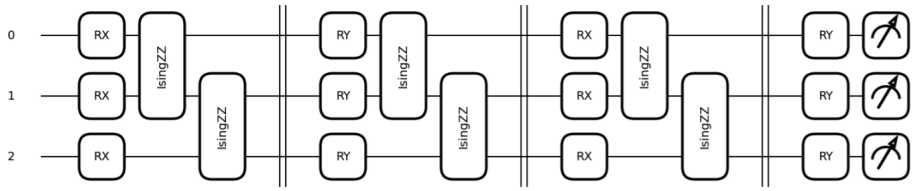

Figure 3: Circuit ansatze for three-qubit Transverse Ising Model with three layers.

Our cost function for this work was the *infidelity* between state $\phi(w, \delta t)$ and and $\psi(w + dw)$

$$L = 1 - |\langle \phi(w, \delta t) | \psi(w + dw) \rangle|^2 \tag{10}$$

This differs, slightly, from the previous work as we did not normalize by our time step, δt^2. However, given that δt is a constant throughout, the final optimized parameters should be the same.

At each time step between 0 and T we optimized our parameters using the standard pennylane 'GradientDescentOptimizer' with a step size of 0.1. For each time step, approximately 40 to 60 optimization steps were needed to converge (Fig. 10).

5 Results

The primary benefit of the original work was to show that pVQD outperforms TVDA as measured by the cumulative infidelity of the optimized circuits for different number of circuit shots. Because we simulated with an exact state vector simulator, our cumulative infidelity is independent of circuit shots and a direct comparison cannot be made at this time.

5.1 Observables

Next we show the results of observables obtained running on different hardware, compared to the "exact" solution. We note that the exact solution provide here was reproduced from data found in the original work. We have not verified the accuracy of the "exact" solution shown here. We have measured the average magnetization along both the x and z directions using the pennylane state vector simulator (Figs. 4,5), IBM's QASM simulator (Figs. 6,7) and IBM's Nairobi QPU (Figs. 8,9).

Utilizing the optimized circuit parameters at each time step, we run the circuit on different hardware and measure our observables. The Jupyter notebook 'run-fit-params-to-get-observables.ipynb' was used to run our optimized circuit on real hardware.

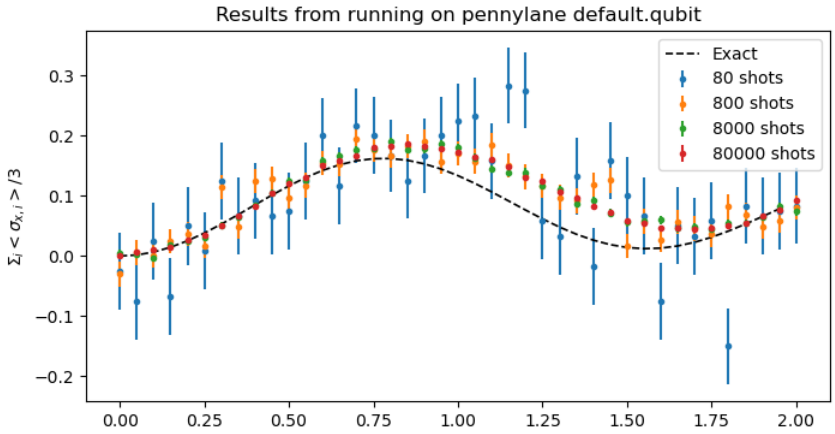

Figure 4: Average magnetization of the three qubits along the direction of the applied magnetic field (x direction). These data were simulated with the pennylane state vector simulator.

5.2 Optimization Steps

Next, for comparison to the original work, we show the number of optimization steps required for each time step. Because the pennylane state vector simulator exactly simulates the quantum state the number of optimization steps when using the simulator is independent of the number of shots (Fig. 10).

We also observed the infidelity for each step of the optimization routine at two different points in time of the simulation, $t = 0.05$ and $t = 2.0$, (Figs. 11, 12)

While we have utilized a slightly different optimization function, the number of steps required to optimize the circuit is comparable to the original work. Future work on this project will likely include the implementation of the exact costs functions implemented in [1].

6 Discussion

Here we have also shown how pennylane facilitates research of this kind with it's built-in tools. This can be clearly seen comparing the code base used here to the original work. Looking at Figs. 4,5 we see that our pennylane simulations models the behavior as in the qiskit version Fig.2 i.e. as we increase the amount

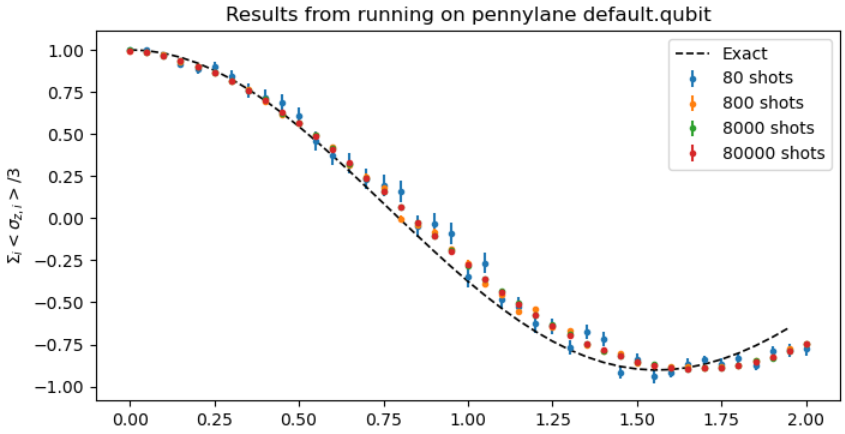

Figure 5: Average magnetization of the three qubits along the z direction. These data were simulated with the pennylane state vector simulator.

Figure 6: Average magnetization of the three qubits along the direction of the applied magnetic field (x direction). These data were simulated with the IBM QASM simulator.

of shots, the curves get closer and closer to the exact solution. This shows that our pennylane version is working as intended and we implemented it correctly. However, one thing to note is that even if we increase the amount of shots to 80,000, the curve doesn't simulate the exact solution completely. There is always a difference as seen in Figs. 4,5 between times t =1 and t = 1.75. We believe this might be because of the way we simulated with an exact state vector and a future direction would be to try and implement the local and global operators as described in [1]. On the flip side, because we implemented the exact state vector directly in pennylane, the time it takes to simulate is reduced by $\sim 100x$ which is a huge improvement! We think this is because of pennylane's efficiency in doing gradient descent rather than qiskit having to import outside ML methods and incorporate them in the code. Now looking at Figs. 6,7 where we run on IBM's simulator that is supposed to model IBM's qpus with noise, we see that the curves are farther away from the exact curve as compared to Figs. 4,5. This is likely due to the noise introduced in the IBM simulator. Finally, we run our pennylane code on IBM's Nairobi machine qpus and this is given in Figs. 8,9. Based on the results of Figs. 8,9, the limitations of simulating the evolution of quantum systems appear to be the errors in the actual QPUs that were available. We find that actual quantum computer are very noisy and the pVQD algorithm doesn't give a good fit in comparison the actual result. We also see that the magnetization at

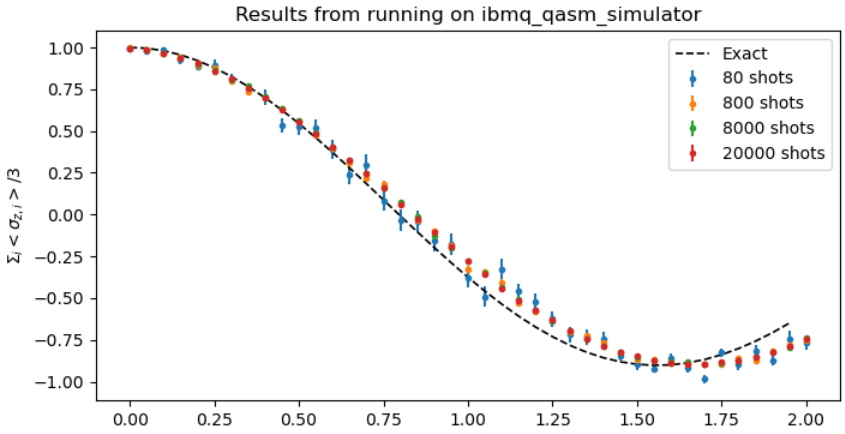

Figure 7: Average magnetization of the three qubits along the z direction. These data were simulated with the IBM QASM simulator.

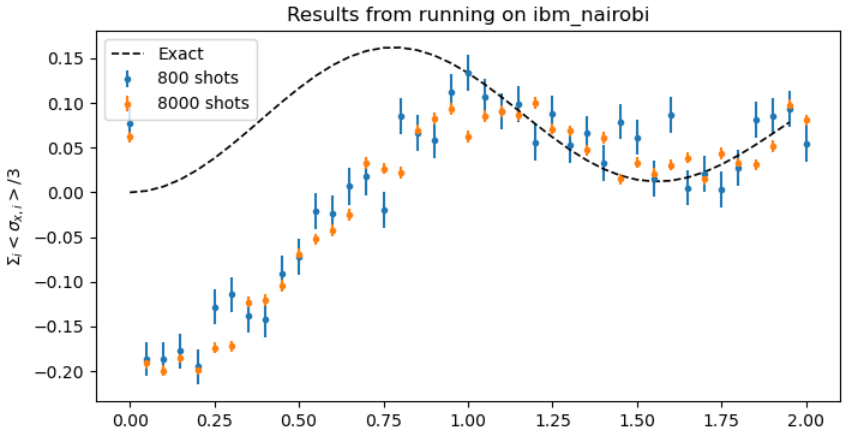

Figure 8: Average magnetization of the three qubits along the direction of the applied magnetic field (x direction). These data were measured with the IBM Nairobi QPU.

the final time step seems to correspond quite nicely to the exact solution so we think there might be some solution bias in this algorithm. A follow-up would be try error mitigation techniques in combination with our pennylane pVQD algorithm.

Another interesting observation is the number of optimization steps required as a function of time (Fig. 10). At larger absolute time values, t 2.0, the number of optimization steps required to minimize the cost function begins to increase. Although there are not enough data to draw conclusions, the increase in the last few time steps appears to increase at a much higher (exponential?) rate. Does this imply, that the state of the system is beginning to change much more rapidly over time, requiring more gradient descent steps to find our new optimal circuit parameters? It's not clear to us that this is the case. At large time values, the change in the total x- and z-components of the qubit spin states are comparable to the change found at earlier times.

6.1 Future Work

Oh man. There are so many future directions in which this work can lead into but we didn't have the time to implement everything we wanted to since this is indeed a hackathon! Additionally, while we were working

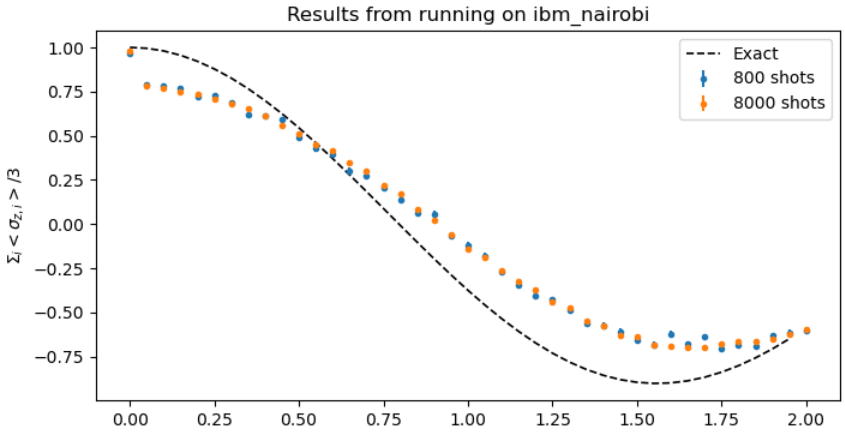

Figure 9: Average magnetization of the three qubits along the z direction. These data were measured on the IBM Nairobi QPU.

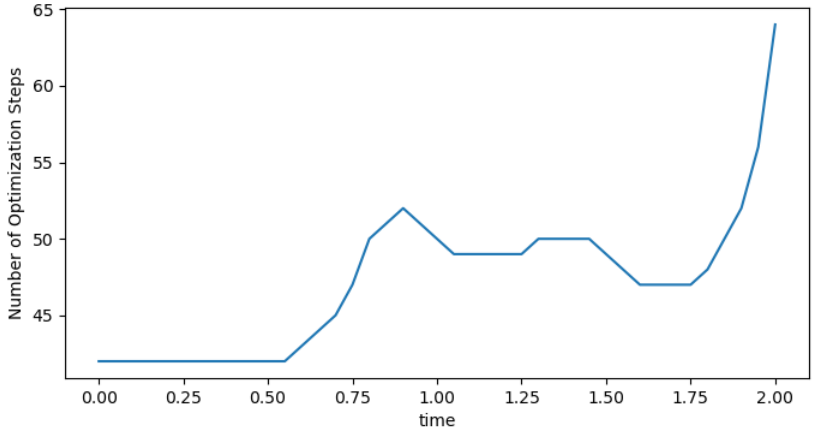

Figure 10: Number of optimization steps required at each time-step of the simulation.

through this project, more and more ideas kept popping up! Here is a list of possible future directions:

1. In respect to the quantum chemistry challenge at QHACK, we can try to simulate the BeH_2 molecule and perform time dynamic simulations of the molecule under external electromagnetic fields.

2. Try other various gradient descent ML schemes with applications to this algorithm to see if we can improve on it.

3. Randomized benchmarking on the average error of pVQD circuits. In a way pVQD is very similar to randomized benchmarking due to way of choosing the circuit ansatz. Since there are in total 24 clifford group operators, perhaps randomized operators for the ansatz might give a better result?

4. Optimize circuit parameters based on the SWAP test, and other figures of merit.

5. Vary the circuit ansatz that they use and see what kind of ansatz would work best for pVQD

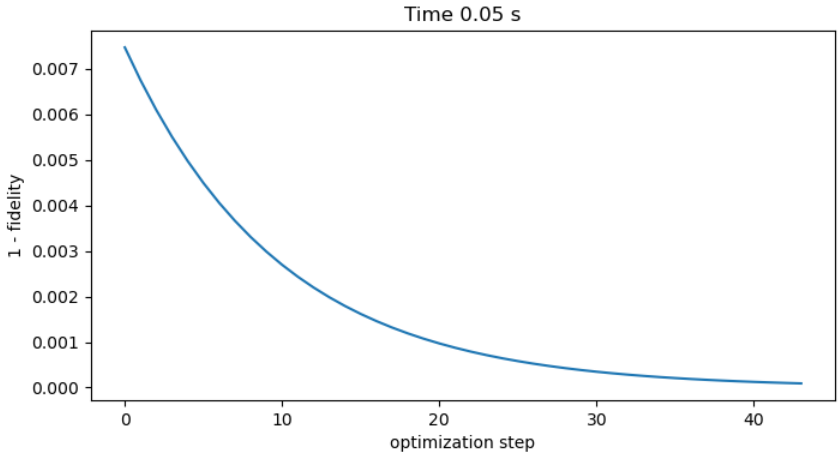

Figure 11: State infidelity at each optimization step at t = 0.05.

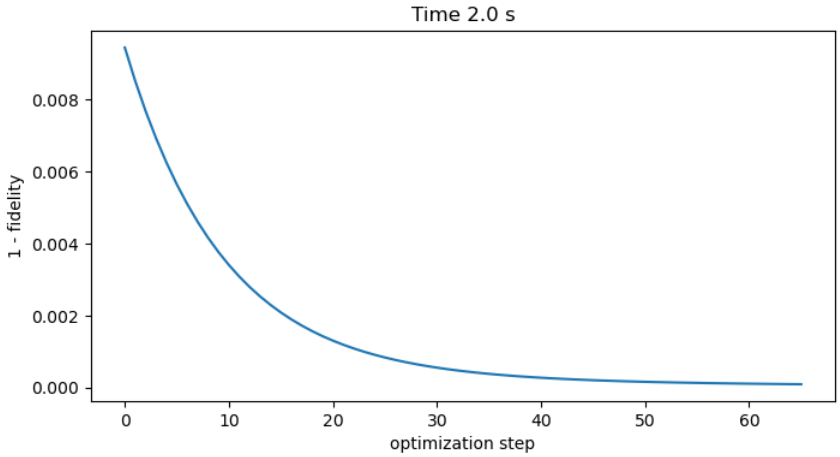

Figure 12: State infidelity at each optimization step at t = 2.0.

References

[1] Stefano Barison, Filippo Vicentini, and Giuseppe Carleo. An efficient quantum algorithm for the time evolution of parameterized circuits. *Quantum*, 5:512, jul 2021.

[2] Ying Li and Simon C. Benjamin. Efficient variational quantum simulator incorporating active error minimization. *Phys. Rev. X*, 7:021050, Jun 2017.

[3] Ollitrault P.J. Tacchino F. et al. Miessen, A. Quantum algorithms for quantum dynamics. *Nature*, 3:25–37, dec 2022.