Academic Year 2022 – 2023
Research Unit: UCLA - MQST
Team(s): Ryan, Arvind, Payal, and Will
Supervisor(s): Richard Ross and Anu Sundararajan

Spring Quantum Algorithms Final Lab Report:
Simulating the Quantum Fourier Transform using Tensor Networks

Ryan Dougherty

June 16, 2023

Abstract

For our Spring quarter lab project, we worked on reproducing the results of the paper *'The Quantum Fourier Transform Has Small Entanglement'* by Miles Stoudenmire, Julien Chen, and Stephen White [1]. In the paper they show that the Quantum Fourier Transform (QFT) has small entanglement which allows for it to be simulated efficiently on a classical computer. With the use of tensor networks, Chen et al were able to represent the QFT as a Matrix Product Operator (MPO) and the input state vector as a Matrix Product State (MPS). Through the use of singular value decomposition (SVD) and an efficient MPO contraction algorithm known as the Zip-Up algorithm, they were able to simulate the QFT efficiently faster than the typical Fast Fourier Transform (FFT) with large input sizes.

My role on this project consisted of implementing the Zip-Up algorithm in Python and using it to prepare the QFT as an MPO efficiently. I will discuss the methods used to implement the Zip-Up algorithm and the results of the simulation. I will cover methods I used to verify my results and discuss approximation methods that can be used to achieve faster runtimes at the cost of minimal error. Lastly, I will also discuss the challenges I faced in implementing the algorithm and the future work that can be done to improve the simulation.

Contents

1 Introduction

1.1 The Zip-Up Algorithm

The Zip-Up algorithm is an efficient algorithm for contracting Matrix Product Operators (MPOs) that scales in runtime complexity of $O(\chi n^2)$, where χ is the largest bond dimension of the final MPO [1]. This time complexity comes from the fact that the intermittent MPOs formed within the algorithm have bond dimensions no larger than χ; this is due to the MPOs having Schmidt coefficients that decay quicker than the final MPO.

1.1.1 Contraction and Stabilization

The zip-up algorithm relies on the methods of performing tensor contraction over small sections of the MPO to guarantee stabilization. Stabilization is a key concept of the zip-up algorithm that I will explain briefly in this section.

The zip-up algorithm performs local SVD at each site when contracting two adjacent MPOs within the circuit. Typically, the resulting singular values from this contraction and SVD *do not* follow the Schmidt coefficients of the contracted MPO. This will lead to a loss of information and accuracy when these singular values get truncated. However, this can be avoided by ensuring the individually contracted tensors between the two MPOs are isometric. This is achieved by performing a series of localized SVD to ensure the contraction is done over the site of orthogonality. This is known as stabilization; the zip-up algorithm over the QFT-MPO guarantees stabilization. For more details on the subject of stabilization, refer to [1, Stoudenmire, Chen, et al; Apendix H].

1.2 Verification

To verify the results of the simulation, I relied on the Payal's implementation of the input Matrix Product State (MPS). I used her calculated MPS in tandem with my calculated MPO to perform a full contraction of the QFT circuit. I then compared the resulting state vector with the state vector obtained from the FFT. I also compared the singular values of the MPO with the Schmidt coefficients of the MPS to ensure the MPO was being prepared correctly. Lastly, I compared the runtimes of the FFT and the zip-up algorithm to ensure the zip-up algorithm was performing as expected.

2 Methods and Implementation

2.1 Code

All of my code was done in Python using a quantum tensor network library called Quimb. All my code can be found at https://github.com/the-iron-ryan/MPO_QFT, where I have included the source Python files for building the QFT-MPO and MPS input states, the zip-up algorithm, and the runtime analysis. I have also included the Jupyter notebook I used to run the simulation and generate the results for this report.

2.2 MPO Preparation with the Zip Up Algorithm

The zip-up algorithm from [1] was implemented in approximately 800 lines of python code. I followed the methods outlined in the paper to implement the algorithm and was able to create a step-by-step diagram of the zip-up for the MPO shown in Figure 1.

I do a series of contractions and SVD at each step, starting from the phase tensors at the bottom left of the QFT circuit. Each of these SVDs will push the singular values to the left side tensor and continue until all the phase tensors are contracted for this local MPO. A 'zip down' process is then done to move the site of orthogonality down to the last phase tensor. As described in section 1.1.1, this is done to ensure that the next contraction will be done on an isometric tensor to ensure minimal Schmidt coefficient error when doing singular value truncation. The process is then repeated until the entire QFT circuit is zipped up into a single column MPO.

With this preperared MPO, I was able to perform a full contraction with the MPS to obtain the final state vector. I then compared this state vector with the state vector obtained from the FFT to ensure the simulation was done correctly.

2.3 Zip Up Algorithm Parameterization

My zip-up algorithm implementation has several important parameters that can be adjusted to alter the overall complexity of the MPO construction and final contraction with the MPS. These parameters include the qubit count, the cutoff threshold (which determines the threshold at which to truncate the singular values when doing localized SVD), and the max bond dimension of the final MPO. I will discuss the effects of these parameters later in Section 3, but for now I will discuss what these parameters mean and how they affect the runtime of the algorithm.

The qubit count is simply the number of qubits within the circuit. This can be adjusted to do a Fourier transform on a smaller or larger amount of data using the QFT MPO with an input MPS of the same size. The cutoff threshold is the threshold at which to truncate the singular values when doing localized SVD. This is a parameter that can be adjusted to achieve faster runtimes at the cost of minimal error. This is due to the low entanglement observed of the QFT circuit observed from the exponentially decaying Schmidt coefficients. Since these coefficients decay exponentially, the localized singular values will decay as well. This means that the singular values can be truncated at a relatively low threshold without losing much information. Lastly, the max bond dimension is the maximum bond dimension of the final MPO. This parameter can be adjusted, but is often fixed at a reasonable value relative to machine precision. In the paper, the authors chose a max bond dimension of 8 for the MPO – which is what I chose for my implementation as well. The source code for the zip-up algorithm can be found in the `zip_up` function inside `MPO.py`. [2]

2.4 Bit Reversal

The last step for preparing the QFT-MPO is to perform a bit reversal on the output MPO after the zip up. This process was at first quite a challenge to uncover, but it turned out to be a simple solution that involves swapping the bonds and site indices. This was accomplished in quimb through a set of `reindex` and `retag` function calls. The details of this swapping can be found in the source code of the `create_MPO` function inside `MPO.py`. [2]

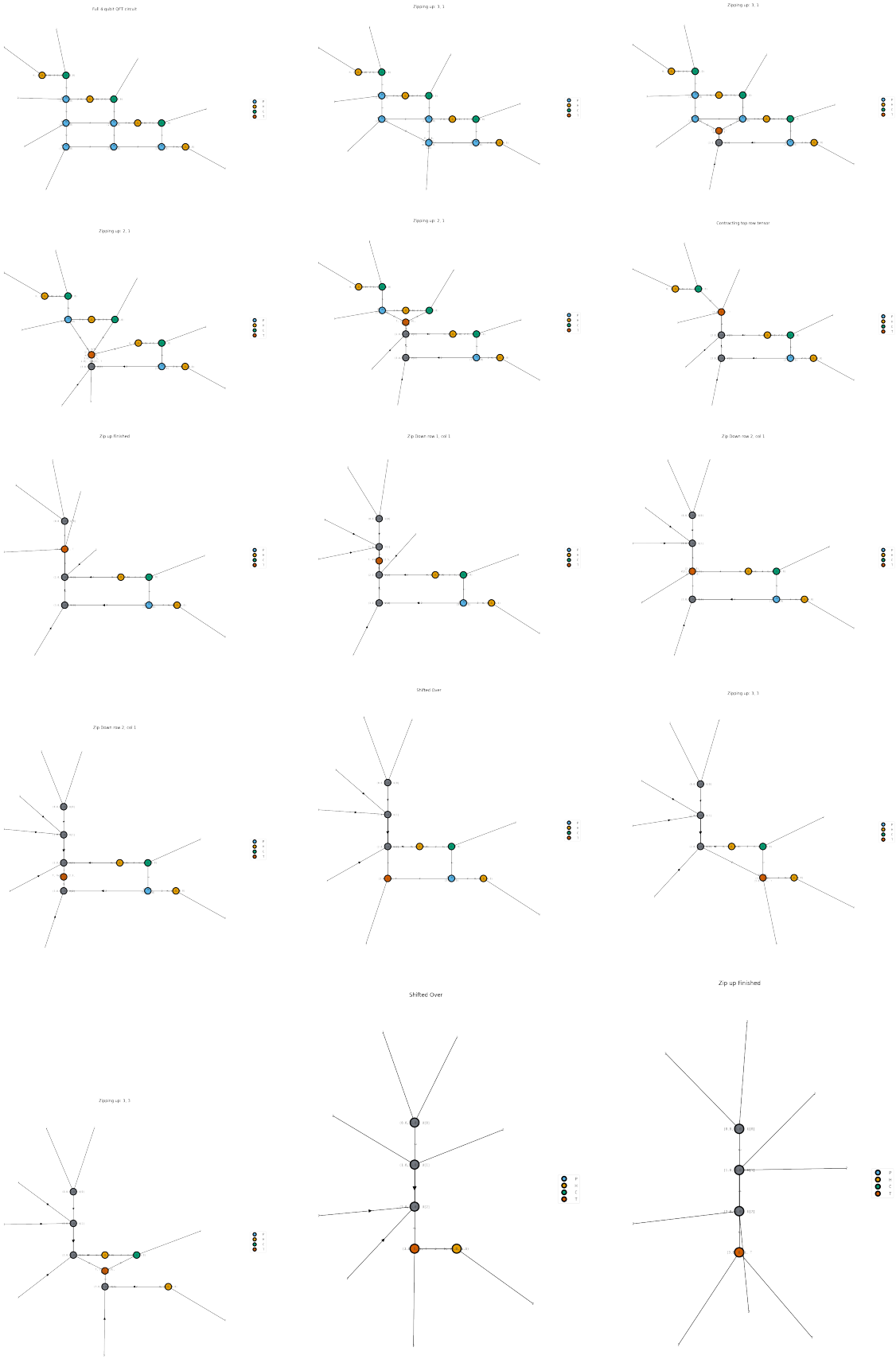

Figure 1 – Diagram of step-by-step 4 qubit zip-up algorithm of the QFT MPO

2.5 Analysis and Benchmarking Setup

I developed code to benchmark the runtime of the QFT-MPO applied to the input MPS. I used the Python timeit library to time the runtime. I also implemented a method to verify the results of the simulation by comparing the state vector obtained from the FFT with the state vector obtained from the QFT-MPO. This was done with a simple difference in the vector norms. This method was used to verify that I was in fact creating the QFT-MPO correctly and that the zip-up algorithm was working as expected.

I did not bother to benchmark the runtime of the zip-up algorithm of the MPO. Mainly because the paper didn't bother to do it, but also because creating the MPO is something that only needs to be done once for a given qubit count. Then is can be reused for each successive input state. Therefore the runtime of the zip-up algorithm is not a concern for the overall runtime of the QFT-MPO.

3 Results and Discussion

3.1 QFT-MPO Runtime Results

3.1.1 Benchmark Functions

The key benchmarks to report were the runtime of the QFT-MPO when applied to an already prepared MPS and the runtime of the QFT-MPO with the MPS state prep time included. The runtime of the QFT-MPO was measured by timing the contraction of the QFT-MPO with the MPS. The runtime of the QFT-MPO with the MPS state prep time included was measured by timing the contraction of the QFT-MPO with the MPS and the time it took to prepare the MPS. I then compared these results to the runtime of the classical FFT algorithm.

I benchmarked my results for the QFT-MPO on three different functions: a Gaussian function, a single cosine function, and a cosine function with a cusp. The Gaussian function and cosine functions were chosen because they are smooth and relatively simple functions to run a Fourier transform on. The cosine function with a cusp was chosen because it is a more complex function that has a slight discontinuity from the cusp. This cusp is a challenge for the QFT-MPO because it requires a large bond dimension to do a spectral analysis of it accurately – which increases runtime complexity. The results of these benchmarks can be seen in Figure 2.

3.1.2 Benchmark Results

Our runtime results of the QFT-MPO alone seem to match the results seen in the paper quite well. We observe an inflection point against the FFT runtime around 18 qubits for all three algorithms, which is exactly what the paper observes. We also see a speed-up on the order or two magnitudes, which is also what the paper observes.

However, we do not see the same runtime results if we include the time needed to construct the MPS. The paper concluded that we would still see at least one order of magnitude better results, but we seem to be seeing results on the same order of magnitude. This is most likely due to different methods of preparing the MPS. The paper uses a method known as Random Singular Value Decomposition (RSVD) to prepare the initial state. We just use a simple method of creating an MPS from a dense input state vector. Refer to Payal's section of the report for more details and insight into better state preparation methods.

3.2 Error Analysis

I also ran an error analysis on my QFT-MPO algorithm. I did this by comparing the state vector obtained from the FFT with the state vector obtained from the QFT-MPO. This was done with a simple difference

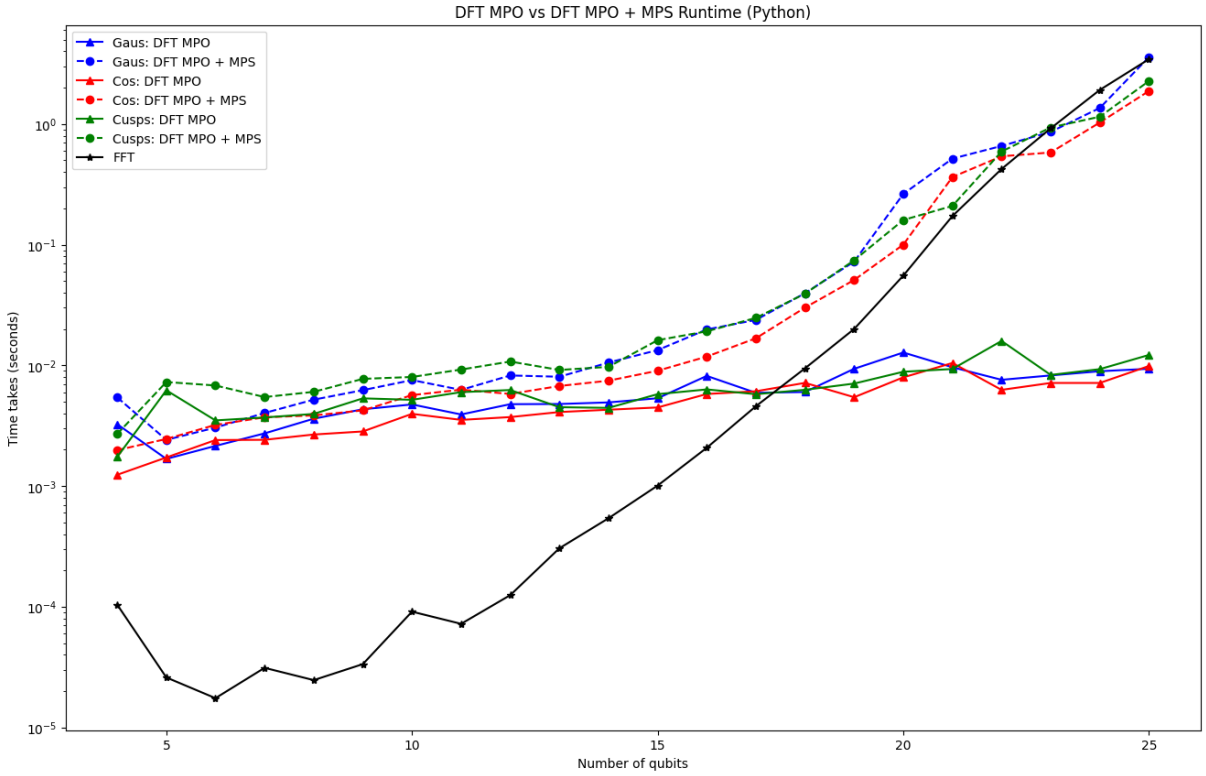

Figure 2 – Runtime results of QFT MPO on three seperate algorithms: Gaussian function (blue), single cosine function (red), cosine function with a cusp (green). These function results are plotted for their QFT runtime (solid line) and their MPS state prep time + QFT runtime (dashed line). Lastly, the state prep and runtime of the classical FFT is plotted (black) for a simple cosine function - since the input function of the FFT has no runtime effect on the algorithm

in the vector norms. I ran this analysis with varying bond dimensions, cutoff thresholds, and qubit counts to see how the error changed.

I chose to vary MPO bond and qubit values that match the tested results in Section III of the paper. [1, Section III] For the first test I fixed the MPO bond size χ to be 4 and varied the qubit sizes to be $n = 7, 8, 9, 10$. For the second test I fixed the qubit size to be $n = 9$ and varied the MPO bond sizes to be $\chi = 5, 6, 7, 8$.

The results of these tests can be seen in Figures 3 and 4. In the first results, we can see that the error decreases relatively linearly with the qubit count, which is to be expected with a fixed bond dimension. In the second results, we can see that the error decreases exponentially with the bond dimension. This result is expected and is a key finding from the paper. The paper shows that the exponentially decreasing error from the bond dimension is due to the exponentially decreasing Schmidt coefficients in the main QFT circuit. This shows that the QFT does indeed have low entanglement and can be efficiently constructed using the zip-up algorithm in $O(\chi n^2)$ time.

It is worth noting that the error reported in the paper is several orders of magnitude lower than the error I am reporting. This is most likely due to the fact that the paper uses a more sophisticated version of the zip-up algorithm and is able to produce a more accurate MPO. However, we do still achieve the same order of magnitude difference in error values between the two tests. This is a great sign that our implementation of the QFT-MPO is working as expected.

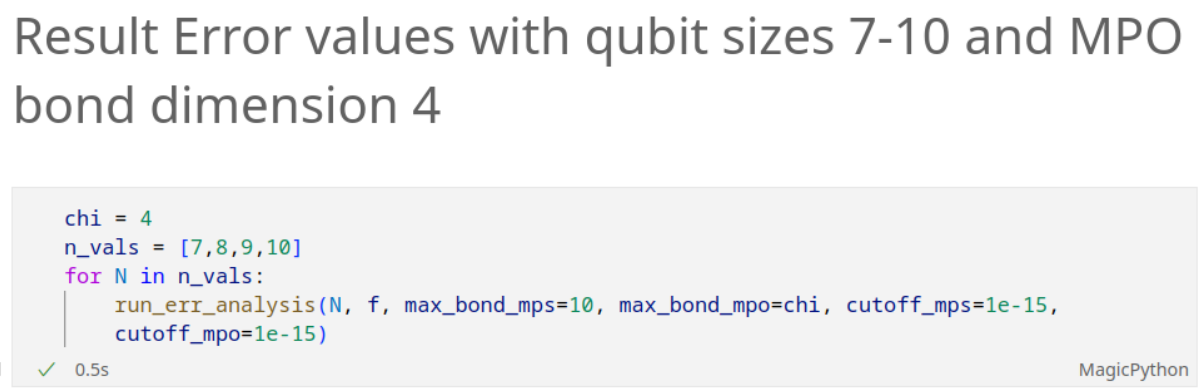

Figure 3 – Error analysis of QFT-MPO with fixed bond size $\chi = 4$ and varying qubit sizes $n = 7, 8, 9, 10$.

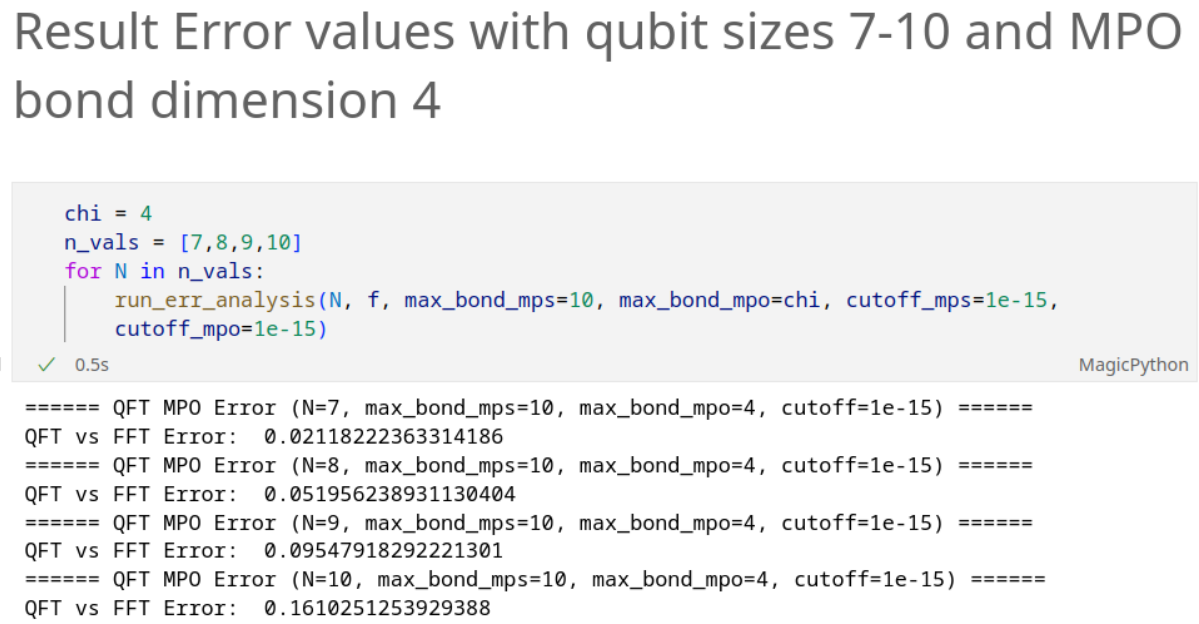

Figure 4 – Error analysis of QFT-MPO with fixed qubit size $n = 9$ and varying bond sizes $\chi = 5, 6, 7, 8$.

4 Conclusion

In conclusion, we were able to successfully implement the QFT-MPO algorithm and verify its results. We were also able to successfully benchmark the runtime of the QFT-MPO and compare it to the runtime of the classical FFT algorithm. We were able to verify that the QFT-MPO algorithm is indeed faster than the classical FFT algorithm for large qubit counts. However, we were not able to verify that the QFT-MPO algorithm is faster than the classical FFT algorithm when we include the time needed to prepare the MPS. This is most likely due to different methods of preparing the MPS. The paper uses a method known as Random Singular Value Decomposition (RSVD) to prepare the initial state. We just use a simple method of creating an MPS from a dense input state vector, which is most likely inefficient compared to the latter.

Lastly, we were able to successfully run an error analysis on the QFT-MPO algorithm and verify that the error decreases exponentially with the bond dimension. This is a key finding from the paper and shows that the zip-up QFT-MPO construction algorithm is indeed efficient and can be constructed in $O(\chi n^2)$ time.

5 Future Work

The low entanglement of the QFT gives rise to additional study into classically simulating other quantum algorithms. If an alogithm's entanglement is low and has exponentially decreasing Schmidt coefficients, then it can be efficiently simulated classically using the zip-up algorithm – or other similar MPO constuction algorithm. This is a very interesting result to verify from Miles, Chen, and Whites' paper and is worth further study.

References

[1] Jielun Chen, E. M. Stoudenmire, and Steven R. White. The quantum fourier transform has small entanglement, 2022.

[2] Ryan Dougherty. *QFT MPO Code*, 2023. Available at https://github.com/the-iron-ryan/MPO_QFT.

[3] Glen Evenbly. A practical guide to the numerical implementation of tensor networks i: Contractions, decompositions and gauge freedom, 2022.

[4] Johnnie Gray. quimb: a python library for quantum information and many-body calculations. *Journal of Open Source Software*, 3(29):819, 2018.

[5] Chase Roberts, Ashley Milsted, Martin Ganahl, Adam Zalcman, Bruce Fontaine, Yijian Zou, Jack Hidary, Guifre Vidal, and Stefan Leichenauer. Tensornetwork: A library for physics and machine learning, 2019.

[6] E M Stoudenmire and Steven R White. Minimally entangled typical thermal state algorithms. *New Journal of Physics*, 12(5):055026, may 2010.