Can You Beat Casey?

Neutral-atom quantum computing is advancing at an incredible pace. The era of analog devices is giving place to gate-based computers and, in particular, error correction protocols. Much of this is due to the amazing compilation design space of this platform, born out of the flexibility of entangling arbitrary pairs of qubits, a high degree of gate parallelization, as well as a wide variety of native gates.

Another reason why quantum computing is advancing at an incredible pace is the folks that have been putting amazing sweat equity in order to advance the field. Today, your main contender is QuEra's Quantum Systems Architect: Casey Duckering. Casey knows QuEra's systems inside and out and was one of the key people to optimize the circuits used in our recent demonstration of magic state distillation with logical qubits [1]. Your goal? To do better!

In this challenge, we will be providing you a set of circuits which may or may not be too wise in design. Even if they look good, they are probably not choosing the best gate set for neutral-atom systems nor ordering operations so that maximum parallelization is possible, particularly given the layout constraints and flexibilities of atom-based devices. On this last point, while arbitrary connectivity is a beautiful asset, it should not be overused: the fewer atom movements, the better the fidelity of a circuit will be. So **we challenge you to use your arsenal of tools and knowledge about circuit design and compilation to optimize these circuits to a specific neutral-atom quantum computing layout while respecting some design rules** that we discuss right below.

Programming infrastructure

We welcome the tides of 2025 with a new version of Bloqade, built on QuEra's also new compiling tool-chain Kirin designed for kernel functions and composable representations. With these advances, Bloqade's infrastructure has been enhanced to be able to represent gate-based quantum computing workflows with a focus on neutral-atom quantum hardware. This includes extending Open Quantum Assembly Language (QASM) programming to accept `annotations`, `for` loops, `if` statements and, overall, all the tooling needed to efficiently represent parallelizable and global gate structures in quantum circuits. Bloqade-circuits remains in its early stages, so expect some bumps as you drive. QuEra's team will be available for roadside assistance!

Challenge statement

- Analyze the circuits below and determine opportunities to rewrite them maximizing parallelism under the hypothetical hardware constraints below
- Write down Bloqade implementations of your circuits and simulate them using the PyQrack simulation module.
- Evaluate the performance of your circuits using our heuristic noise model and depth calculator via Bloqade

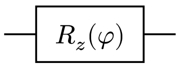
Design Rules

We have a list of references that can help you understand some details of neutral-atom gate-based devices [1], [2], [3], as well as some neat tricks for circuit rewrites [4]. You can access these for free on the open-access repository arXiv. Use these sparingly, as the time is tight for the challenge. You can also use Quirk as a prototyping tool to help you think through circuit deformations/transformations. Furthermore, you may want to investigate using Bloqade's or other third-party compilers to create a pipeline to pre-process circuits into better formats before fine-tuning choices specific for the neutral atom architecture you will be targeting below.

Here goes the summarized set of design rules we will use for our challenge. When in doubt, all options and possibilities defer to the constraints defined here.

Native Gate Set

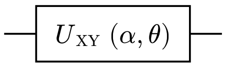
Neutral-atom quantum computers have some flexibility with regards to their native gate set, and a given choice depends on the specific hardware implementation. We will consider the processor to be functionalized in two areas: (i) a storage zone and (ii) an entangling zone (details below). Building on that, we will consider the following scenario:



Z-rotation

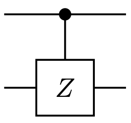
Z-rotations can be done globally (i.e. on all qubits for the same angle) or locally (i.e. on a sub-selection of qubits, with the same angle for all qubits in this subset). The choice of global or local rotations imparts different errors in the qubits, with global rotations being strongly favored. In fact, global Z-rotations are substituted for simple delays in phases of subsequent gates, so they are free of errors and cost effectively zero time.

Global **and** local 1-qubit gates can be done without moving the atoms, and we will assume that can affect them both in the storage and entangling zones. Read more about our zones and geometric constraints below.



XY-rotation

Similar to the Z rotations above, we can do rotations around any axis in the XY-plane both globally or locally. These rotations are defined by an arbitrary angle α on the plane, setting the rotation axis at an angle α in reference to the x-axis, with the rotation angle defined by θ. Once more, global operations are favored over local ones.



The standard entangling gate for us will be the controlled-Z gate. To apply the entangling gate, atoms have to be put in proximity of each other. This is why the gate zone is structured the way it is, with pairs of sites to enable the gate to occur between atoms. The ability to shuttle atoms means you can pick any two atoms in the storage zone regardless of their distance apart and move them to the proper pair of sites in the gate zone. Furthermore, you can pick up multiple atoms and put them in the gate zone *in parallel.*

Importantly, we will assume here that every time a set of parallel CZ operations are applied, the corresponding atoms will be moved to the entangling zone, and then moved back to their respective original positions. **No shuffling of register address will be allowed.** Your performance will be tied to your ability in leveraging parallelism in circuit design.

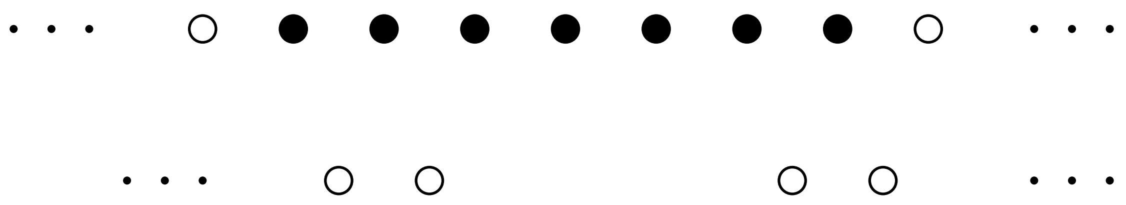
Native gates and Bloqade

The set of gates above can be summarized via general and parallelizable U3(θ, φ, λ) (defined as per usual convention as in here) gates, and our trusty CZ. This gate set can be accessed on Bloqade via `parallel.u` and `parallel.cz`, primitives that are able to incorporate lists of targets to be deployed jointly as needed. Note that Bloqade also has an automated rewrite capable of converting any circuit to this gate-set. That can be accessed as `from bloqade.qasm2.rewrite.native_gates import RydbergGateSetRewriteRule`; an example of using it for automated transpilation is provided in our exemplary tutorial materials based on GHZ state preparation. Look for it on the challenge folders.

Register Geometry

To keep things simple, in this challenge we will consider a simplified version of a neutral-atom quantum processor. It will be defined by two zones: a **storage zone** with atoms about $4\mu m$ apart, and a **gate zone** with pairs of $20\mu m$ apart. The distances are not terribly important for this challenge. Both of them are chosen to be 1-dimensional straight lines.

Here is an illustration to help you make the description above concrete:

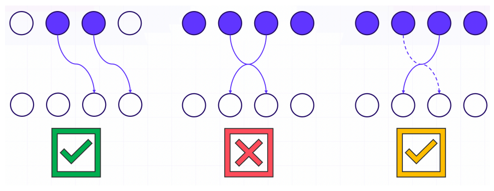


The first row with black dots correspond to the storage zone and contains your atoms. Once you define the number of qubits in your register, a line segment of host spots *will* be filled with qubits; atoms in this zone can go through 1-qubit rotations, either local of global. The second row with pairs of white spots correspond to the gate zone. Atoms from the storage zone have to be moved to these white spots when you want to entangle them. After that, they move back to the storage zone. Once registers have been defined, we will assume (unrealistically) infinite extensions of the zones to either sides of the original register. This will keep the spreading of moves between zones simple for automatic compilation and error modeling.

Shuttling Dynamics

Despite not being allowed to choose geometric addresses in this challenge, the dynamics of moving atoms does affect the actual performance of your program from a parallelization perspective. Here are the two simple rules which constraining your atom-moving strategies:

1. A single move operation may pick up any subset of atoms from one row and then place those atoms in another row (or somewhere else in the same row) under the constraint that the atoms are not reordered during the move and all atoms end up in the same destination row. Shortly, the spots that pick up and drag atoms while they move cannot cross paths. Crossings can be done in series though, but that costs more moves.



2. Atoms are not allowed to collide. A spot with one atom cannot get a second one.

Error budget

Showcasing the features of Bloqade and of good, hardware-aware, circuit design implies being able to compare performances of different deployments. In this challenge, you will be judged directly by the performance of your final circuits, as much as by the quality of your exposition on your thinking process.

For our purposes, Bloqade already contains channels for Pauli errors and atom losses. These can be applied randomly to a circuit and their effect can be tracked by sampling iterations of operations. From those, fidelities or KL divergences can be computed. An example follows in the GHZ tutorial you will find in the hackathon folder.

We will work with a simple hierarchy of errors:

$$E_{\mathrm{CZ}} > E_{1-\mathrm{qubit},L} > E_{1-\mathrm{qubit},G}$$

That is, CZ errors - which in our current specific setting include pick/drop errors from atom moves, induce losses,etc - are worse than local 1-qubit gates These are then worse than global 1-qubit gates. Furthermore, decoherence is taken into account and an incentive exists to parallelizing your circuit as much as possible, reducing its total run time.

In sum: entanglement costs the most, and optimizing moves should come as the second priority. Then, global gates are favored over local ones.

Evaluation criteria

The performance of the different teams on this challenge will be evaluated through a few different criteria. In order of priority and value, these are:

- the validity of solutions with respect to hardware constraints and with respect to the expected circuit for the solution.
- values obtained by **fidelity** estimation relative to the heuristic noise model in Bloqade.
- the utilization of Bloqade's features for circuit generation such as kernels and control flows
- the suite of tools and pipelines developed by participants to aid and/or automatize solutions.
- the finding *and reporting* of bugs via issues on Bloqade's Github
- creativity of approach and results.

These criteria begin quantitative and become more qualitative as we go down the list. Quantitative analyses will take precedence over more qualitative ones, the latter being used for tie-breaking.

Challenge Circuits

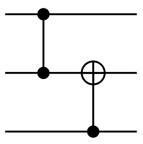
While you can manually input the circuits below into any SDK of your choice for further analysis or optimization, note that for the purposes of judging (and for your own use!) there are QASM files that correspond to each circuit in the `assets\qasm` folder, numbered to align with the order of circuits below.

You may also consult the `assets/examples` folder which contains examples of preparing the GHZ state using bloqade-circuits.

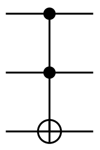
Warm-up circuit set

The following will NOT count towards your score but are left here to encourage you to think about how one could go about optimizing your circuit.

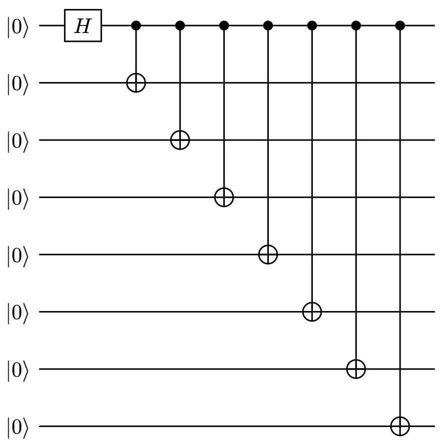
Let's start with a simple warm-up, with the two circuits below counting towards your scoring. First, let's just look at something unrealistic. Try to optimize the circuit below for our native gate set and minimal number of layers:

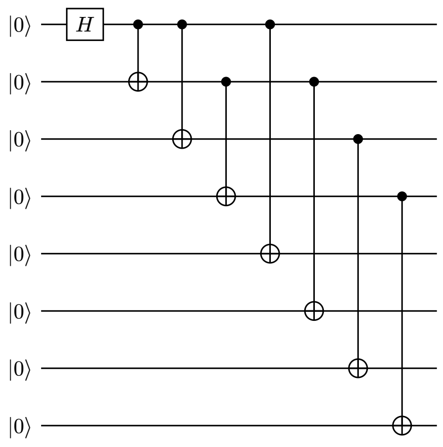


Now, let's add one more simple circuit to this warm-up exercise. Break a Toffoli gate into an efficient native circuit for our system:



Then, take a look at the following two circuits. They are equivalent circuits for preparing a Greenberger-Horne-Zeilinger (GHZ) state but one is more optimal than the other for the architecture given to you. Try to figure out why!





Look for the Bloqade tutorial notebook in our Github repo for an in-depth analysis and implementation of this GHZ example.

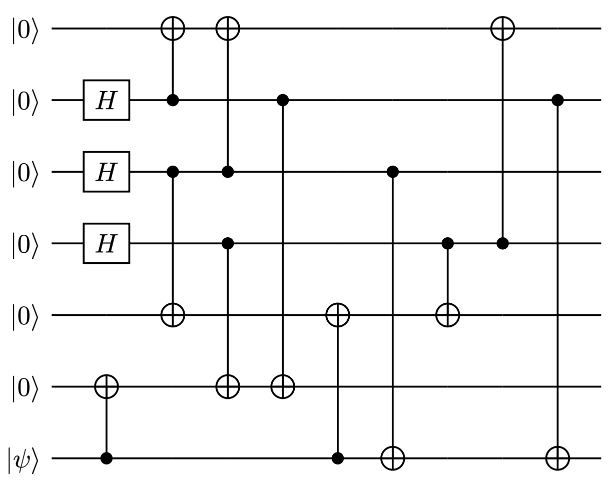
Finally, let's look at a quantum error correction (QEC) code. We can stick with encoding a single qubit into Steane 7-qubit $[[7,1,3]]$ code (also known as the smallest color code!). The $[[n,\ k,\ d]]$ notation indicates that 7 physical qubits are used to encode 1 logical qubit of "distance" 3, with the relation $d = 2t + 1$ telling us that this code can correct any error on a singular $(t = 1)$ qubit. This is the shorter, simpler encoding used in the recent logical magic state distillation work by QuEra [1].

The goal is to take an arbitrary state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, and encode it in logical states as $\left|\overline{\psi}\right\rangle = \alpha\left|\overline{0}\right\rangle + \beta\left|\overline{1}\right\rangle$, where

$$\begin{aligned}\left|\overline{0}\right\rangle &= \frac{1}{\sqrt{8}}(|0000000\rangle + |1010101\rangle + |0110011\rangle + |1100110\rangle \\ &\quad |0001111\rangle + |1011010\rangle + |0111100\rangle + |1101001\rangle) \\ \left|\overline{1}\right\rangle &= \frac{1}{\sqrt{8}}(|1111111\rangle + |0101010\rangle + |1001100\rangle + |0011001\rangle \\ &\quad |1110000\rangle + |0100101\rangle + |1000011\rangle + |0010110\rangle)\end{aligned}$$

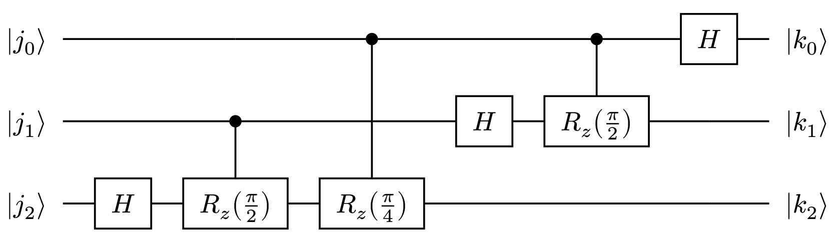
When doing error correction, we usually omit the logical states, but we show them here to keep things concrete. More information about this encoding and its intuition for operation can be found in the original paper by Shor here

The encoding circuit is as follows:



More information about this encoding can be found here and here.

Circuit #1
Now let's go to something more realistic. Let's optimize a standard quantum Fourier transform circuit:



For an idea of a non-trivial implementation, draw inspiration from this paper.

Circuit #2
Scaling difficulty! QAOA on a graph. Assume you have the following graph:

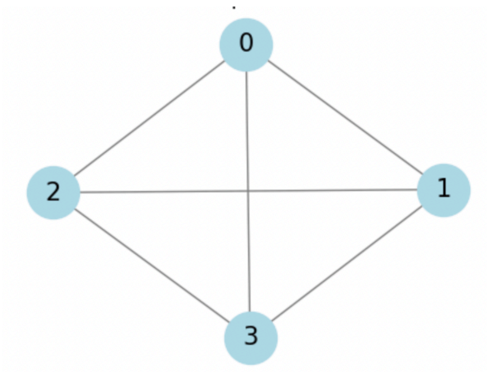
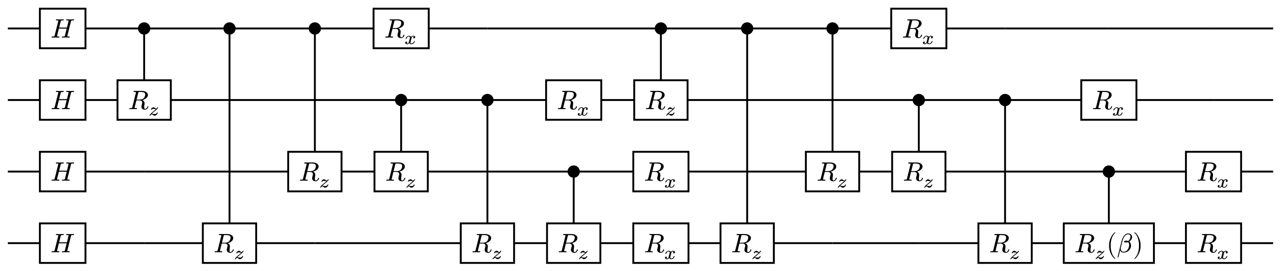


Figure 2: A 4-vertex 3-regular graph.

We build a simplified QAOA protocol for MaxCut on top of it as:

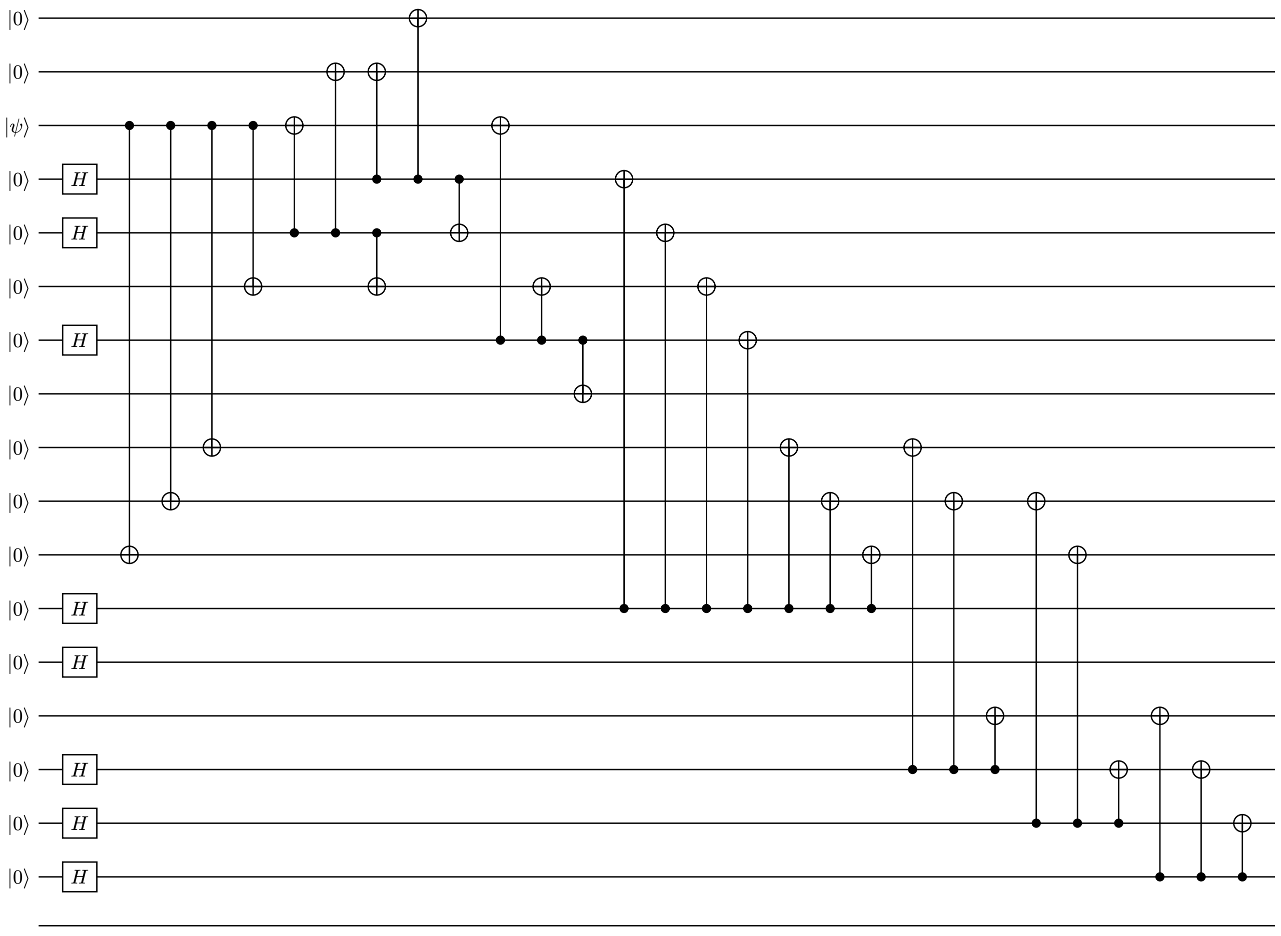


All $R_x = R_x(\alpha)$ and $R_z = R_z(\beta)$. The exact angles used in this circuit and which you are to be scored on are provided in the Cirq program in the `assets` folder on Github, under `QAOA_generator.ipynb`. This program can generate random 3-regular graphs and a simplified QAOA protocol for MaxCut on top of the graph. In this case, the above circuit and graph were obtained with the default arguments of `seed=1` and number of qubits `N=4`. A possible idea to improve parallelization here is to consider "Hamming weight phasing" as a strategy [5],[6].

Your are encouraged, although this is not required for minimal scoring, to change the parameters to create larger circuits or an ensemble of graphs and try out your optimization skills!

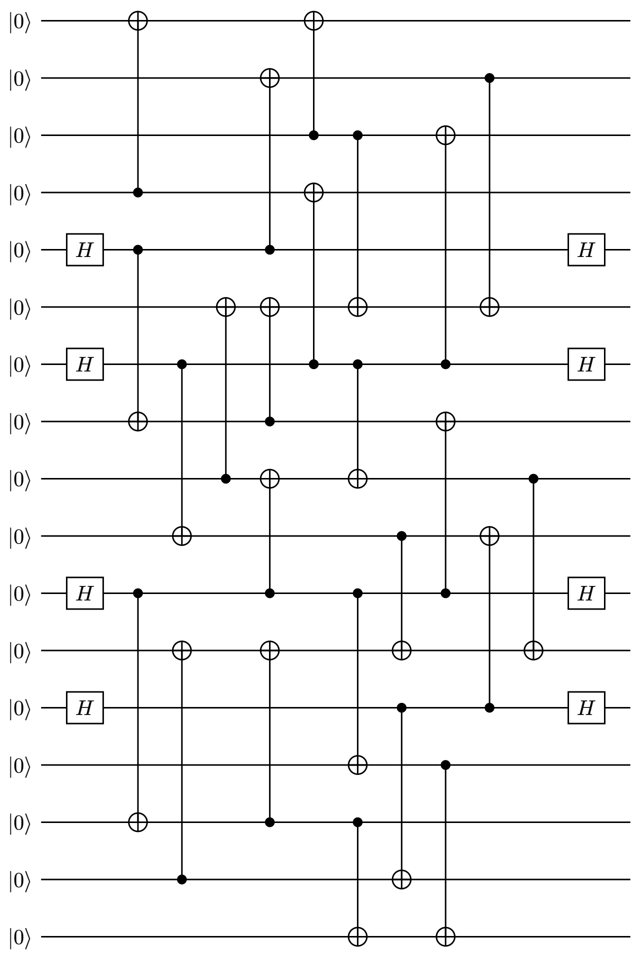
Circuit #3

Let's now revisit the color code example. This time, do a 17-qubit encoding that brings us to [[17,1,5]] extension of Steane's code. Here goes an un-optimized version of it, for an arbitrary state $|\psi\rangle$ to be encoded into the logical qubit:



Validating changes in this circuit can get pretty hard. You can try and prove that your optimization is both valid and improves the original circuit. Happy optimization!

Circuit #4
Sticking the QEC theme, your final challenge is the surface code (encoding just the 0 state here):



Bonus circuit
Feeling bold? Got tired of our ideas? BYOC! (bring your own circuit!) Flex your optimization muscles and showcase how neutral atoms can be used for your favorite quantum protocol, deploying it on Bloqade!

Extra Activities

We will be welcoming side activities that improve visualization, argumentation, or creative pipelines that get you to your solution faster. While these may not count directly to the performance of your cost function, you may want to pursue such directions in order to help you argue that your choice is indeed optimal or better than those of your competitors. These activities may count in case of the need for tie-breaking between competing teams. As alluded throughout this document, here are some ideas:

- Build Quirk analyses of your circuit optimization process
- Utilize third-party compilers (such as Enola or ZAC) to generate animations of your atom moves

- Modify the noise model in order to analyze the effects that different hardware improvements would bring to your circuit

Bibliography

[1] P. S. Rodriguez *et al.*, "Experimental Demonstration of Logical Magic State Distillation." [Online]. Available: https://arxiv.org/abs/2412.15165

[2] D. Bluvstein *et al.*, "A quantum processor based on coherent transport of entangled atom arrays," *Nature*, vol. 604, no. 7906, pp. 451–456, Apr. 2022, doi: 10.1038/s41586-022-04592-6.

[3] D. Bluvstein *et al.*, "Logical quantum processor based on reconfigurable atom arrays," *Nature*, vol. 626, no. 7997, pp. 58–65, Dec. 2023, doi: 10.1038/s41586-023-06927-3.

[4] W. Schober, "Extended quantum circuit diagrams." [Online]. Available: https://arxiv.org/abs/2410.02946

[5] C. Gidney, "Halving the cost of quantum addition," *Quantum*, vol. 2, p. 74, Jun. 2018, doi: 10.22331/q-2018-06-18-74.

[6] I. D. Kivlichan *et al.*, "Improved Fault-Tolerant Quantum Simulation of Condensed-Phase Correlated Electrons via Trotterization," *Quantum*, vol. 4, p. 296, Jul. 2020, doi: 10.22331/q-2020-07-16-296.