Ventura's Quantum Associative Memory, or QuAM, is an algorithm made of two parts: storing the information into quantum registers and using Grover's to search for the desired value. The quantum advantage offered increases storage capacity exponentially, although this space efficiency comes with a cost in terms of time. Recovering a saved pattern requires running Grover's algorithm, which takes $O(2^n)$ time. Classically, for m binary patterns of length n, known algorithms range from requiring mn bits(storing set of patterns in a lookup table or RAM), to only being able to store less than or equal to kn patterns, for some constant $0.15 \leq k \leq 0.5$, both of which offer linear storage capacity, $O(n)$. In this paper, we review an algorithm that offers a storage capacity of $O(2^n)$ while using n neurons. We also note that using gates and code from class, where the algorithm required $2n + 1$ qubits, we only use $n + 2$ qubits, albeit at another cost in terms of time. Our multi-qubit control gates require more gates to work compared to their multi-qubit controls, which take longer.

We first implement the algorithm of storing m binary patterns of length $n$by figuring out what the given gates correspond to. The given algorithm calls for 3 registers, an $x$register with n qubits, one for each bit in the pattern, a $g$register with $n - 1$qubits, a garbage register to temporarily identify a certain state, and a $c$register with two control qubits, acting as our auxiliary qubits to identify which patterns to change and which patterns are saved. However, we found that the $g$register was not necessary at all and instead use our Toffoli gates that can check the values of our c register with the signature. Thus, we use $n - 1$fewer qubits and improve the scalability of the QuAM algorithm. For storing our patterns, we first apply a FLIP gate to turn our initial 0 states into our first pattern. We then split our state into a superposition of a "larger" piece and a "smaller" piece with our S operator. Finally, the SAVE operator simply controls on the *x* register, finding the pattern that we just created and performing a controlled-NOT on the c_1

qubit, which marks it as a completely saved state. We then repeat this algorithm $m$times, until we have our patterns all in equal superposition. Our FLIP gate. We are given a matrix for S_p, where p iterates down from *m* to 1 as we save each successive state. Naively, when you plug in small values, 1, 2, and 4, you get nice multiples of $\boldsymbol{\pi}$ for your unitary. However, for $p = 3$, no nice multiple of sinusoidal functions equal $\sqrt{2}/\sqrt{3}$. Thus, instead, we simply give our θ a value of $arccos(\sqrt{(p - 1/p)})$, and our S_p operator is a controlled unitary gate with θ,0,0,0 as our values, controlling and targeting on our c registers. Finally, the SAVE operator simply controls on the *x* register, finding the pattern that we just created and performing a controlled-NOT on the c_1 qubit, which marks it as a completely saved state. By the end of these iterations, the *g* and *c* registers are all separable from the *x* register, so we can ignore them and read the state as an equal superposition of the patterns given to save.

After we have the state saved in the *x* register, we have the task of performing a recall of a certain bitstring. This can be done by using Grover's algorithm, which requires implementing a unitary that matches the given bitstring. However, because we don't start with an equal superposition of every state that the normal Grover's algorithm requires, we need to make a slight modification to the second iteration of Grover's. Rather than applying the unitary that applies a phase rotation to our target state, we instead apply a phase rotation to every state that was originally saved in the superposition. To do this, we control on the x register, identifying each originally saved pattern and applying the phase rotation. After this step, all of the non-target states have the same phase, and the target state has a higher magnitude. Then, we continue a normal Grover's algorithm for the final R - 2 iterations, continuing to amplify the magnitude of the target state until we can measure it.

In the paper, the authors make this rotation gate seem highly trivial, and not one sentence is used to discuss the barriers to implementation. In fact, what we find is that there is virtually no way to differentiate which states are stored in our pattern to which states are merely an effect of superposition. Our efforts of using auxiliary qubits, quantum phase estimation and QFT were all to no avail. In fact, the most promising solution, quantum phase estimation, only returns a phase of a unitary gate. However, our stored patterns may or may not be in an entangled superposition of our x register, and thus, there is no way to represent that state as multiple unitaries. This problem remains to be solved quantumly in an efficient manner. Instead, we store our implementation classically stores our patterns and apply a phase kickback protocol to flip the phase of our stored states. We note that there exists a quantum version of this, but it is in fact less efficient as you have to store the bits as quantum gates instead.

Finally, we put our two sub-algorithms together to produce QuAM. The only addition to QuAM is the ability to search our pattern given some partial pattern, which is easily implemented by taking in unknown values as question marks, saving the indexes in which we have values, and only applying Grover's to our registers that have values. We implement this algorithm successfully. However, we notice one drawback of the algorithm. In the case that you have a query with known bits that correspond to more than one stored pattern, our probability of detecting a correct pattern decreases dramatically. In fact, when there is only 1 pattern, we get an amplitude of roughly 900/1024, corresponding to 88%, whereas with 2 possible patterns, the probabilities are both around 250/1024, which adds up to under 50%. This in fact comes from us not being able to know the value of M, the known solutions, in Grover's without classically parsing the array. One final note is that our associative memory collapses at the instance of Grover's search, which defeats the purpose of effective memory. This problem can be solved

easily with an implementation similar to 3-qubit repetition from quantum error correction. By creating a new circuit and CNOTing all our qubits, we can create a dummy circuit to measure and retrieve our stored pattern while still retaining our original memory.

To conclude, the quantum associative memory algorithm allows us to efficiently store a series of bit patterns and recall them given full or partial patterns. This can find applications in implementing quantum arrays and hashmaps which can be used to efficiently store long bit patterns in logarithmic space efficiency. Some drawbacks to the algorithm come in the form of time costs, as well as inefficiencies when searching for partial patterns with multiple matches due to the number of iterations in Grover's algorithm.