Quantum Random Access Memory For Dummies

Koustubh Phalak
CSE Department
Penn State University
PA, USA
krp5448@psu.edu

Avimita Chatterjee
CSE Department
Penn State University
PA, USA
amc8313@psu.edu

Swaroop Ghosh
School of EECS
Penn State University
PA, USA
szg212@psu.edu

***Abstract*—Quantum Random Access Memory (QRAM) has the potential to revolutionize the area of quantum computing. QRAM uses quantum computing principles to store and modify quantum or classical data efficiently, greatly accelerating a wide range of computer processes. Despite its importance, there is a lack of comprehensive surveys that cover the entire spectrum of QRAM architectures. We fill this gap by providing a comprehensive review of QRAM, emphasizing its significance and viability in existing noisy quantum computers. By drawing comparisons with conventional RAM for ease of understanding, this survey clarifies the fundamental ideas and actions of QRAM.**

***Index Terms*—Quantum Computing, Quantum RAM, Qudit, Bucket-brigade QRAM, Flip-flop QRAM, EQGAN, PQC**

I. Introduction

Quantum Computing (QC) has progressed rapidly in the past decade. With the advancement in qubit technologies such as superconducting qubits [1], trapped ion qubits [2], photonic qubits [3], quantum dots [4] and diamond nitrogen-vacancy centers [5], implementation of quantum algorithms on quantum computers has become practically possible. This has also enabled the application of quantum computing in various fields such as machine learning [6], finance [7], chemistry [8], cybersecurity [9], and advanced manufacturing [10]. A potential game changer in quantum computing is the augmentation of Quantum Random Access Memory (QRAM) that has shown a potential to provide exponential speedup for algorithms such as, Fourier transform [11], discrete logarithm [12], and pattern recognition [13]–[15]. QRAM is also a key requirement for important quantum algorithms such as quantum searching of classical database [16], [17], collision finding of hash and claw-free functions [18] and distinctness of elements in a list [19], [20]. Along with this, a QRAM can also serve as an important memory element to load classical data into the quantum Hilbert space, compared to simpler methods like amplitude, angle, and basis embeddings [21].

Existing literature on QRAM fails to summarize key aspects of QRAM and explain them in layman's terms which is the objective of this paper. In [22], the authors discuss various QRAMs such as bucket-brigade QRAM, large width small depth QRAM, and small width large depth QRAM but from a fault-tolerance standpoint rather than a fundamental explanatory perspective. An overview of the practicality of QRAM in modern NISQ systems is provided in [23] however, it can be esoteric at times to fully comprehend. We provide a simple-to-grasp review of QRAM for readers interested in diving deep into the field of quantum memories. While complex mathematical knowledge of quantum physics is not required, we do assume that the readers know the fundamentals of quantum computing [24] such as the ket notation, quantum gates, and quantum circuit notation.

This paper is organized as follows: In Section II, we provide preliminaries on quantum computing and the workings of classical RAM. In Section III, we delve into the fundamentals of QRAM, answering key questions about its structure, utility, and requirements. Section IV explores the practical implementation of QRAM, while Section V offers an overview of the challenges in implementing QRAM and its future potential. Finally, we conclude in Section VI.

II. Preliminaries

A. An Overview of Quantum Computing

*a) **Qubits**:* Qubits, the elementary units of quantum computing, are distinct from classical bits in that they can exist in a superposition of states and represent both 0 and 1 simultaneously. This unique property allows quantum computers to perform multiple computations in parallel, providing the potential for exponential speedup compared to classical computers. In a Hilbert space, a qubit is represented by a two-dimensional vector denoted as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β the coefficients of the basis states of a qubit. These coefficients are constrained by the normalization condition $|\alpha|^2 + |\beta|^2 = 1$, and the probabilities of measuring the state of the qubit in the basis state of $|0\rangle$ or $|1\rangle$ are given by $|\alpha|^2$ or $|\beta|^2$ [24].

*b) **Quantum Gates**:* Quantum gates are the fundamental operations that act on qubits in a quantum circuit, akin to how classical logic gates operate on classical bits. These gates include the Pauli-X, Pauli-Y, Pauli-Z, Hadamard, CNOT, and Toffoli gates. They are often depicted as unitary matrices that act on qubit states, maintaining the quantum properties of the system [25]. Quantum gates are created and realized physically utilizing a variety of techniques, including lasers, magnetic fields, and microwave pulses [24].

*c) **Quantum Circuit**:* Quantum circuits are collections of quantum gates that work together to carry out particular quantum computations. Initialization of the qubits is the first step of a quantum circuit. Gate operations, such as multi-qubit gates like the CNOT gate and the Toffoli gate, as well as single-qubit gates like the Hadamard and Pauli gates, are

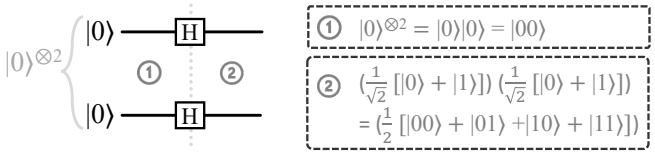

Fig. 1. The presented circuit illustrates a fundamental instance of quantum superposition. It commences with an initial 2-qubit state, ①, and culminates in a superposition state, ②, demonstrating the essential properties of quantum systems. ①: $|0\rangle\,|0\rangle = |00\rangle$; ②: $\frac{1}{2}[|00\rangle + |01\rangle + |10\rangle + |11\rangle]$.

used to change the qubits to the required state. Prior to execution, the high-level gates in the circuit, including the Toffoli gate, are disassembled into a native gate set of the quantum hardware (called transpilation). The output of the quantum circuit is then obtained by measuring the qubits using a measurement gate, which converts the quantum state into a classical state [26].

*d) **Quantum Entanglement**:* When two or more qubits are correlated in a way that prevents one qubit from being described independently of the other qubits, this phenomenon is known as quantum entanglement. This characteristic is critical for the development of effective quantum algorithms and protocols, including quantum teleportation and superdense coding [27]. The non-local correlations of entanglement highlight its importance in quantum computing by allowing the execution of tasks that are classically impossible.

*e) **Quantum Superposition**:* Superposition is a phenomenon that allows both the computational basis states $|0\rangle$ and $|1\rangle$ to exist in quantum Hilbert space at the same time. A qubit state can be put into superposition using Hadamard (H) gate. If the initial qubit state is $|0\rangle$ ($|1\rangle$), then the superposition state becomes $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ ($\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$) after the H gate. We present an example in Fig. 1 to generate a superposition of all the basis states for a two-qubit system.

*f) **Quantum Algorithms and Applications**:* The potential of quantum computing has been illustrated by a number of quantum algorithms. Examples include Grover's method for exploring unsorted databases [16], Shor's algorithm for factoring large integers [12], and the quantum simulation algorithms [28]. Among other applications, these algorithms have substantial effects on cryptography, optimization, and quantum system simulation. Combinatorial optimization issues can be resolved using variational quantum-classical algorithms such as, Quantum Approximate Optimization Algorithm (QAOA) [29], Variational Quantum Eigensolver (VQE) [30] and Quantum Machine Learning (QML) models like Quantum Support Vector Machine (QSVM) [31] and quantum principal component analysis (QPCA) [6].

B. Classical RAM

Desktop computers can often slow down when running data-intensive applications. To address this issue, one solution is to install additional Random Access Memory (RAM), which can provide a temporary storage medium for the central processing unit (CPU) to retrieve data quickly in any order while executing a program. It is a volatile 'read/write' memory that stores data temporarily while the computer is operational.

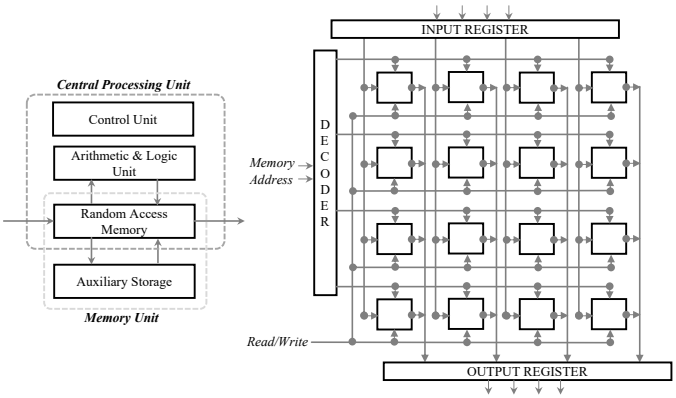

Fig. 2. ***Left:*** Depiction of the placement of RAM within the memory hierarchy, highlighting its proximity to the CPU. The RAM's speed can be attributed to this closeness, as it serves as an intermediary between the CPU and auxiliary memory systems. ***Right:*** A detailed representation of the various functional components within a RAM, illustrating their organization and interconnections. *Memory Array:* It is made up of a grid of rows and columns that stand in for memory cells used to store data. One piece of information is stored in each cell. *Input Register:* During a write operation, it temporarily stores the data that will be stored in the memory array. *Output Register:* During a read operation, it temporarily stores the data that was read from the memory array. *Decoder:* This component takes the memory address from the address bus and converts it into row and column coordinates so that it can access the associated memory cell. To demonstrate the flow of row and column signals, arrows from the decoder should point in the direction of the memory array. *Control Bus:* This transmits read and write enable signals to the memory array to control data access activities.

When the computer is switched off, the stored data is lost due to its volatile nature. RAM is more efficient than hard drive storage for temporary storage due to faster access time. The fundamental capability of any computing device is the ability to store and manipulate information in a series of memory cells organized in an array [32]. RAM is the most well-known architecture for such a memory array since it allows each cell to be addressed [33].

A memory array, an input register, and an output register constitute RAM. The memory cells are organized into rows and columns, with each cell holding one bit of data. Data is accessed and manipulated using address lines, data lines, and control lines (read and write enable signals). When the CPU needs to access data from the memory, it sends the memory address through the address lines. Depending on the read or write signal, the data is either retrieved from the memory cell (read operation) or stored in the memory cell (write operation) [34]. The contents of a memory cell are recovered and transferred to the output register when the address of that cell is loaded into the address register (a procedure known as 'decoding'). A traditional RAM requires effective data storage, retrieval, and manipulation in order to function. The two main types of RAM, Static Random Access Memory (SRAM), and Dynamic Random Access Memory (DRAM), have unique characteristics that determine their use in different applications [35]. Fig. 2 illustrates the position of RAM within the memory hierarchy and presents a functional block diagram showcasing its key components and their interactions.

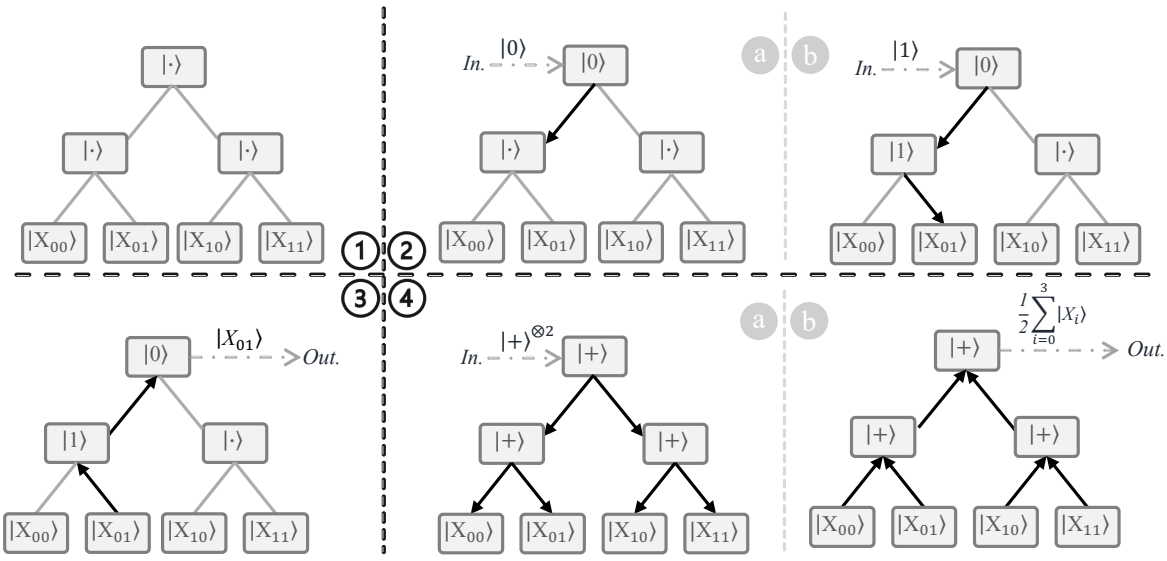

Fig. 3. Working of a bucket-brigade QRAM with 2 address lines and 4 memory cells. ① Initial state of the QRAM, all quantum switches are initialized to $|\cdot\rangle$ state which is a waiting state where the quantum switch waits for incoming qubit states of the memory address to be accessed. ② Input register activates switches for output register to access data in address $|01\rangle$ ($|X_{01}\rangle$). The address qubits are sent in a sequential top-down fashion starting from the Most Significant Bit (MSB) all the way to the Least Significant Bit (LSB). In the example shown, first MSB qubit $|0\rangle$ is sent that changes the state of the root quantum switch ⓐ, followed by the LSB qubit $|1\rangle$ that routes the switch to the direction of the memory cell $|X_{01}\rangle$ ⓑ. ③ Output register reads data $|X_{01}\rangle$ via route of activated quantum switches. ④ Superposition of all addresses turning on all quantum switches ⓐ to read superposition of all the data ⓑ. Note that $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle) + |1\rangle)$., *In.*: Input register, *Out.*: Output register.

TABLE I
A COMPARISON OF CLASSICAL RAM AND QUANTUM RAM.

Attributes	Classical RAM	Quantum RAM
Information storage	Classical bits $(0/1)$	Qubits $(\lvert\psi\rangle = \alpha\lvert 0\rangle + \beta\lvert 1\rangle)$
Access mechanism implementation	Using transistors and capacitors	Encoding into superposition
Read operation	Read signal	Quantum swap operation
Write operation	Write signal	Qubits in input register
Gate activations	$\Theta(2^n)$; $n = \#$bits	$\Theta(n)$; $n = \#$qubits
Error correction	Repetition codes	Surface codes
Scalability	Increasing #bits	Increasing #qubits

III. FUNDAMENTALS OF QRAM

A QRAM is a memory element analogous to RAM that is able to store data in quantum format. Similar to RAM, a QRAM has three components, the input (or address) register, the output (or data) register, and the memory arrays. The difference here though is that the input and output registers are composed of qubits instead of bits, while the memory arrays can be either classical or quantum depending on the usage of QRAM [36]. For example, for the two fanout QRAM implementations in [37] the optical implementation has 1-bit classical memory cells that change the polarization of the output register photons based on the bit value, while the phase gate implementation uses two superconducting qubits in a single memory cell (one for storing information, one for extracting information). Table I shows the difference between RAM and QRAM. Another key difference in QRAM is the way memory access is performed. Rather than accessing a single memory location at a time, QRAM uses superposition to simultaneously access multiple memory locations. This is made possible by leveraging the power of quantum Hilbert space, where all memory addresses are first loaded into superposition. The overall state is then passed through the QRAM to obtain another superposition state, this time with addresses and data combined. Let's say that we have n qubits and consequently $N = 2^n$ address lines. All the addresses will be represented as basis states, from $|0\rangle$ to $|N-1\rangle$ [36], and are stored in address register r. Each address $|i\rangle$ will have amplitude α_i, so the effective superposition of addresses will be $\sum_{i=0}^{N-1} \alpha_i |i\rangle_r$. This superposition state is then sent to QRAM and the output is another superposition state, which contains both the address state and the data state picked from data register o. If X_i is the data in address i, then the output state of QRAM is $\sum_{i=0}^{N-1} \alpha_i |i\rangle_r |X_i\rangle_o$. Effectively, the storage of data in QRAM can be summarized through the following equation

$$\sum_{i=0}^{N-1} \alpha_i |i\rangle_r \xrightarrow{QRAM} \sum_{i=0}^{N-1} \alpha_i |i\rangle_r |X_i\rangle_o$$

However, retrieving the data can be challenging due to the no-cloning theorem [38]. This is generally taken care of by performing entanglement operations between memory cell qubits and output register qubits using gates such as SWAP gate or CNOT gate.

With the aforementioned statements, readers may become curious and ponder over several crucial aspects of QRAM, namely: (i) the motivation behind the need for QRAM: *Why do we need a QRAM?* (ii) the configuration of QRAM: *What is the structure of a QRAM?* and (iii) the extent of QRAM's utility and its usage: *Where is a QRAM used?* In the following sub-sections, we provide answers to these questions.

A. Why do we need a QRAM?

In quantum computing, the fundamental building blocks of computation are quantum states, which can represent information as a superposition of basis states. These quantum states are fragile and sensitive to external disturbances, such as environmental noise and decoherence [39], which can cause them to rapidly lose coherence and become unusable for computation. Therefore, it is crucial to be able to efficiently store and retrieve quantum states themselves in order to execute quantum algorithms.

Classical memory devices are not suitable for storing quantum states since they will require collapsing of the wavefunction with a measurement operation [40]. The collapse of the wavefunction destroys the superposition of states and causes the quantum state to take on a singular classical value (either 0 or 1), which can be stored in classical RAM but no longer be valuable for quantum computation. QRAM is a potential solution to this problem as it allows quantum states to be stored and retrieved efficiently without collapsing the superposition of states. This is accomplished by using quantum mechanics to encode information in a way that is resistant to decoherence and other sources of noise [41]. This allows quantum states to be stored and retrieved with minimal error, making QRAM an essential component of quantum computing technology.

QRAM can also be potentially useful for loading classical data into quantum Hilbert space. Hybrid quantum-classical optimization algorithms in the field of QML often require the conversion of classical data in Euclidean space (e.g., image datasets like MNIST, Iris, CIFAR-10/100, etc.) to quantum Hilbert space into quantum states. This is achieved using encoding methods such as angle embedding, amplitude embedding, and basis embedding [21]. Amplitude embedding embeds 2^n classical features on n qubits, while angle and basis embeddings embed n classical features on n qubits. A problem with these methods, however, is that they are rather simplistic in nature and do not take the complexity of the dataset into account. A QRAM-based loading of data can potentially address the above issue.

B. What is the structure of a QRAM?

Various flavors of QRAM architectures have been proposed as described below:

a) ***Bucket-Brigade QRAM****:* The very first proposal of a QRAM [36] implemented a bifurcation graph-based structure as compared to the traditional d-dimensional lattice of memory arrays (shown in Fig. 2). It is called bucket-brigade QRAM, and the bifurcation graph for this QRAM is a binary tree with the leaf nodes as the memory cells, and the rest of the nodes as switches to route the address state to the correct cell. Overall, there are three main components of this QRAM: the *input register*, the *QRAM itself*, and the *output register*. Note, input/index/address register and output/data register/quantum bus are used interchangeably in the literature. For ease of understanding, we use the terms input and output registers in this paper. There are primarily two cases to explain how a bucket-brigade QRAM works. First, is when there is only a single address in the input register, and second, is when there is a superposition of addresses in the input register. We explain them in the following paragraphs:

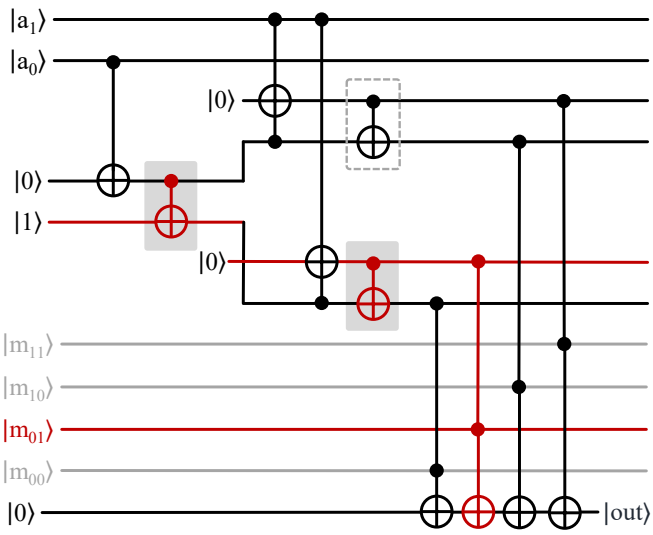

Fig. 4. Circuit-based implementation of a bucket-brigade QRAM. Data in memory cell m_{01} with address $|01\rangle$ is being accessed via a series of CNOT and Toffoli gates performing intermediate computation on ancilla qubits. Note that the CNOT gates highlighted in red are the ones getting activated and the red path represents the active route of the QRAM.

Single address case: Let's consider a QRAM that supports two addresses (two qubits) and four memory cells for storage. The bifurcation graph for this QRAM at the start is shown in Fig. 3, with the quantum switches initialized at the wait state, and the four memory cells present at the leaf nodes. Each quantum switch is a three-level system with states $|\cdot\rangle$, $|0\rangle$, and $|1\rangle$ unlike a qubit which is a two-level system ($|0\rangle$ and $|1\rangle$). This three-level system is often referred to as a qutrit and is inspired by the classical three-level system trit, which are generally tri-state logic multiplexers [42]. The significance of the wait state $|\cdot\rangle$ in each quantum switch is that whenever a qubit state (either $|0\rangle$ or $|1\rangle$) is received by the switch, it changes from $|\cdot\rangle$ to the received state. This helps to achieve that the next time the same switch receives another qubit state, it will route the qubit state to one of its children's node switches. The wait state ensures that un-accessed memory cells are not disturbed. The direction of routing depends on the state of the qubit. Typically $|0\rangle$ ($|1\rangle$) routes the next state to the left (right) child.

Next, we show the example of an incoming address $|01\rangle$ accessing the initialized QRAM. The address state comes from the input register in a sequential fashion from the Most Significant Bit (MSB) to the Least Significant Bit (LSB). Since the address is $|01\rangle$, the MSB is $|0\rangle$ and the LSB is $|1\rangle$. Therefore, state $|0\rangle$ is first sent to the root node switch of the QRAM. Since the root node switch is in the $|\cdot\rangle$, it changes state to $|0\rangle$ (Fig. 3.2(a)). Next, the LSB state $|1\rangle$ arrives at the root node switch which routes it to the left child. The left child is then activated to state $|1\rangle$ (Fig. 3.2(b)). In this way, all the address qubits in the input register are used to create a route to memory cell $|X_{01}\rangle$. Along with the

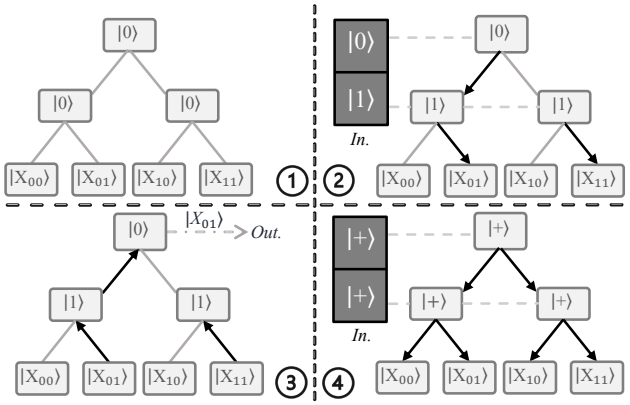

Fig. 5. Working of a fanout QRAM. ① Fanout QRAM initialization with all quantum switches initialized to $|0\rangle$ state. ② Address qubits in the input register controlling the state of their respective quantum switches and creating a path from root node switch to the desired memory cell X_{01}. ③ Data in memory cell X_{01} being accessed in output register via the active route of switches. ④ Superposition of addresses switching on all routes to access contents of all memory cells.

graph-based implementation, we also show the circuit-based implementation of the bucket-brigade QRAM with two address lines and four memory cells (inspired from [43]) in Fig. 4.

After the creation of the route, the data in the output register can either be stored or read in the routed memory cell. In Fig. 3.3, we show the example of reading contents of address $|01\rangle$ ($|X_{01}\rangle$) from the memory cell to the output register via the route. Note that if one wants to store new data, the direction of routing will be opposite i.e., from the output register to the memory cell.

Superposition of addresses: In this case, all the qubits will be present in a superposition state similar to the case shown in Fig. 1. When a qubit in superposition encounters a quantum switch, the switch will also change from $|\cdot\rangle$ to superposition state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Because the superposition state has the presence of both $|0\rangle$ and $|1\rangle$ state, the quantum switch ends up activating both the left and right routes. In this way, when all the superposition address qubits arrive on all the quantum switches, the routes to all memory cells get activated (Fig. 3.4(a)). Next, when the read operation on the output register is performed, contents from all the memory cells traverse from all the activated routes and load a superposition of all the data $\frac{1}{2}\sum_{i=0}^{3}|X_i\rangle$ on the output register. Compared to a classical RAM, the advantage provided by bucket-brigade QRAM is that for n address lines the classical RAM requires O(2^n) transistor activations for accessing data in a single address, while the QRAM takes only O(n) quantum switch activations. Furthermore, at a comparable classical cost of O(2^n) quantum switch activations, the QRAM can read data from all the addresses.

*b) **Fanout QRAM**:* A follow-up paper [37] on the original bucket-brigade QRAM work [36] presents architectural implementations of bucket-brigade QRAM along with another QRAM termed 'fanout' QRAM. Fanout QRAM is taken directly from its classical equivalent fanout RAM, where k^{th} address bit controls 2^k switches. Usually, for n-bit binary address, the MSB is considered 0^{th} address bit and the LSB is considered $n-1^{th}$ address bit. The quantum version of the fanout RAM has k^{th} address qubit controlling 2^k quantum switches. A difference between the quantum switches of bucket-brigade QRAM and the fanout QRAM is that while bucket-brigade QRAM requires qutrits, the fanout QRAM requires only a two-level system so qubits are used as the quantum switches. Initially, all the quantum switches are initialized to $|0\rangle$ state. Next, the address qubits present in the input register are used to change the state of the quantum switches. All the quantum switches connected to an address qubit in state $|0\rangle$ will remain at state $|0\rangle$ while the ones connected to an address qubit in state $|1\rangle$ will change their state to $|1\rangle$.

TABLE II
CREATION OF DATASET WITH ROTATION ANGLE FOR FF-QRAM.

Address (A)	**Data (X)**	**Data Value**	**Normalized Value (X_N)**	**θ = 2arcsin(X_N)**
00	x_{00}	2(10)	$2/\sqrt{15} = 0.51$	1.06
01	x_{01}	3(11)	$3/\sqrt{15} = 0.77$	1.74
10	x_{10}	1(01)	$1/\sqrt{15} = 0.25$	0.5
11	x_{11}	1(01)	$1/\sqrt{15} = 0.25$	0.5

To explain the functionality of the fanout QRAM, we once again consider a QRAM with two address lines and four memory cells. The bifurcation graph for the QRAM with the quantum switches initialized to $|0\rangle$ state is shown in Fig. 5.1. The input register like the previous case can contain either a single address or a superposition of addresses.

Single address case: Let us first take the simple case of single address access, where the memory cell in address $|01\rangle$ (X_{01}) is being addressed. The MSB address state $|0\rangle$ has index 0 and it will control $2^0 = 1$ quantum switch, which is the root node switch. The LSB address state $|1\rangle$ has index 1 so it will control $2^1 = 2$ quantum switches which are the two child switches of the root node switch. The root node switch will stay at state $|0\rangle$ while the child switches will change state to state $|1\rangle$ (Fig. 5.2). As a consequence of this, all the quantum switches are activated, but only a single complete path from the root node to the memory cell (X_{01} in this case) is active. After this, the contents from the memory cell are either updated from or loaded into the output register via the active path (Fig. 5.3).

Superposition of addresses: Extrapolating the above process to a superposition of addresses, the quantum switches will also be switched to superposition. As a result of superposition, each quantum switch will activate routes towards both of its child switches. We show this in Fig. 5.4. Finally, for a read operation, contents from all the memory cells will traverse all the active routes and the output register will read a superposition of data (similar to Fig. 3.4(b)). Compared to the bucket-brigade QRAM, the fanout QRAM activates O(2^n) switches for both single address access as well as for accessing the superposition of all addresses.

*c) **Flip-Flop QRAM**:* A more recent quantum circuit-based QRAM implementation called flip-flop QRAM (FF-QRAM) has been proposed in [44]. The FF-QRAM stores

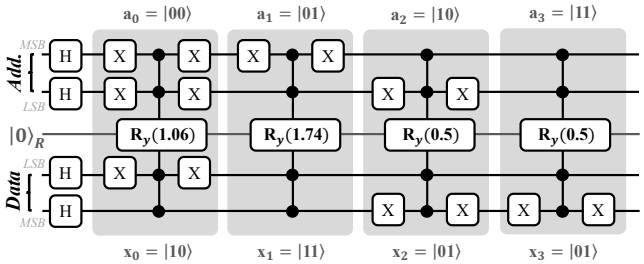

Fig. 6. Working of FF-QRAM circuit. The QRAM stores data 10, 11, 01 and 01 in addresses 00, 01, 10 and 11 respectively.

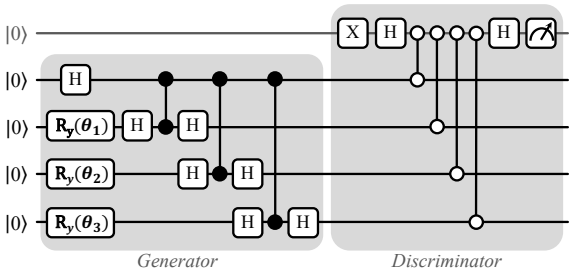

Fig. 7. EQGAN variational QRAM circuit for storing superposition of data from class 0. For storing the superposition of data in class 1, the generator ansatz is slightly different.

binary data in superposition one by one, such that the overall circuit has exponential circuit depth and linear width in terms of the number of address lines (or address qubits). Let's assume there are n address lines and the size of each binary data in an address is m bits. Then, the QRAM circuit will have circuit depth O(2^n) and circuit width O($n+m$). Storing a single data point occurs in three stages: the *flip* stage, the *register* stage, and the *flop* stage. The flip stage is a 'compute' stage that is used to make all the data and address qubit states to $|1\rangle$ that is being stored, the register stage consists of a multi-controlled rotation gate that stores the data in a register qubit, and the flop stage is an 'un-compute' stage which performs the inverse operation of the compute stage on the address and data qubits.

To explain the working of FF-QRAM, consider a two-address line QRAM with four address-data pairs with each data point of size 2 bits ($n = 2$; $m = 2$). For each of the four addresses, we have the data as shown in Table II, and for each data, we generate its respective rotation angle in two steps, (i) we normalize the data. Given we have the data $\{2, 3, 1, 1\}$, the normalization factor will be $\sqrt{2^2 + 3^2 + 1^2 + 1^2} = \sqrt{15}$, and we divide the dataset with the normalization factor, so new normalized dataset becomes $\frac{1}{\sqrt{15}}\{2, 3, 1, 1\} = \{0.51, 0.77, 0.25, 0.25\}$. (ii) We compute the rotation angle for each data using the normalized values. The rotation angle θ_k for a datapoint x_k with normalized value $x_{k,n}$ is given as $\theta_k = 2\arcsin(x_{k,n})$ [45]. For the example shown in Table II, the rotation angles will be $2\arcsin(\{0.51, 0.77, 0.25, 0.25\}) = \{1.06, 1.74, 0.5, 0.5\}$.

Once we have the rotation values computed, we can build the FF-QRAM circuit for these address-data pairs. We show the FF-QRAM for our example dataset in Fig. 6. The quantum circuit will have 2 qubits for address lines, 2 qubits for data lines, and 1 qubits for register lines. First, all the address and data qubits are initialized to $|0\rangle$ state and put into superposition using Hadamard gate. After this, the process of storing the data starts. As mentioned earlier, the data is stored one by one in three stages. In the flip stage, which is the compute stage the qubit states of the relevant address and data are flipped to $|1\rangle$ so that the multi-controlled R_y rotation gate is triggered to store the rotation of the desired data. This is achieved using classically controlled-NOT gates [44]. Basically, when the classical bit value is 0 the NOT gate is activated otherwise the NOT gate is not activated. We present a simpler version of this gate for ease of understanding as follows: when the classical bit value is 0, we place an X gate and when the classical value is 1, we do not place the X gate. Consider the first address-data pair $|00\rangle - |10\rangle$ in Fig. 6. For address qubits, since both the target address lines are in state $|0\rangle$, we put an X gate on both the qubits. Similarly, for the data lines, we put an X gate only on the LSB qubit. Next is the register stage where we add the multi-controlled R_y gate with the computed rotation angle of 1.06 radians. Finally, we then add the inverse of the flip stage in the flop stage, which will be X gates placed only on those qubits where X gates were placed in the flip stage. The same process is repeated for all the remaining address-data pairs as shown in Fig. 6. The end result of this repetitive process is that the FF-QRAM will have a superposition of addresses and their corresponding data and angle stored in the address, data and register qubits, respectively.

*d) **Entangling Quantum Generative Adversarial Network (EQGAN) QRAM**:* EQGAN [46] is pure quantum entanglement-based Generative Adversarial Network (GAN) which is PQC-based and has a quantum generator and a quantum discriminator. They both are trained together with a minimax game. For discriminator model D with parameters θ_d, generator model G with parameters θ_g, real data σ and generated data $\rho(\theta_g)$, the minimax problem is given as

$$\min_{\theta_g} \max_{\theta_d} C(\theta_g, \theta_d) = \min_{\theta_g} \max_{\theta_d} \{1 - D_\sigma[\theta_d, \rho(\theta_g)]\}$$

This EQGAN model is used for variational QRAM as an application, where data points from two Gaussian distributions are stored. The QRAM [46] uses two generators with exponential peak ansatz, one for class 0 and one for class 1 (each class signifies data from one Gaussian distribution), and a swap-test-based discriminator. We show the circuit of this variational EQGAN QRAM in Fig. 7 which stores data from class 0. For class 1 the PQC is nearly the same, with a slight difference in generator ansatz. Using this generator-discriminator setup, in constant O(1) gates the QRAM is able to store data into superposition approximately. Another advantage of this QRAM is observed in a classification task. Without QRAM, training the data on a Quantum Neural Network (QNN) yielded an average 45% classification accuracy, and with the QRAM augmented, the average classification accuracy increased to around 65%.

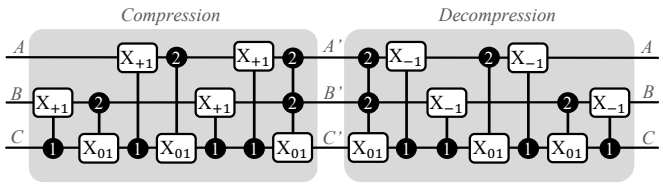

Fig. 8. Compression and decompression circuits for a 2-3-1 compression scheme [48]. The compression circuit compresses the contents of three qubits (A,B,C) in two qutrits (A',B') and generates a free ancilla in state $|0\rangle$ (C'). The decompression circuit again reverts the qutrits and ancilla back to the original three qubits.

*e) **Qudits-based memory**:* Qudits are higher-state quantum units that contain more than two computational basis states. While a qubit in superposition can be represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, the superposition in a qudit with d computational basis states is represented as

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle + ...\alpha_{d-1}|d-1\rangle = \sum_{i=0}^{d-1} \alpha_i|i\rangle$$

Recent works such as, [47], [48] propose qudits-based quantum memory where qubits are compressed onto qudits temporarily in their higher states using reversible compression circuits. When unused, the qudits can be used as ancilla bits somewhere else. During computing, the qudits can be reverted back to qubits by performing the inverse operation of the compression circuit.

The authors in [48] define two higher state gates analogous to the X-gate. Let's assume that the input qubit state is $|i\rangle$, then (i)X_{+t} gate performs the operation $|i\rangle \xrightarrow{X_{+t}} |(i+t) \bmod d\rangle$ and (ii) X_{ij} gate performs $|i\rangle \xrightarrow{X_{ij}} |j\rangle$ and $|j\rangle \xrightarrow{X_{ij}} |i\rangle$. There are also the two-qudit controlled versions of these gates where the control qudit has a reference control state and the target qudit has the gate. They also introduce an x-y-z qudit-qubit compression scheme that is dependent on the radix of the input and output qudits. Here, x is the radix of the input qudits, y is the radix of the output qudits and z are the number of ancilla generated. The compression scheme should follow $x^a \leq y^b$, such that $0 < b < a$ and $a - b = z$, where a,b are integers. x^a denotes the number of computational basis states of the input and y^b denotes the number of computational basis states of the output. A natural restriction is that the computational basis states of the output should be higher than that of the input. Another restriction is that the number of input qudits a should be greater than the number of output qudits b for enabling compression. As a result of this compression, a total of $a - b = z$ ancilla qubits are generated.

A simple example is the conversion of qubits ($d = 2$) to qutrits ($d = 3$). Three qubits can store $2^3 = 8$ computational basis states, and two qutrits can store $3^2 = 9$ computational basis states. So three qubits can be compressed into two qutrits and the leftover qubit can be generated as an ancilla that can be used in other circuits. For this example, we have $x = 2$, $y = 3$, and $z = 1$, so it is a 2-3-1 compression scheme. We show the compression and decompression circuits of this scheme in Fig. 8. The compression circuit consists of controlled X_{+1} and X_{01} conditioned either on $|1\rangle$ or $|2\rangle$ states. The decompression circuit has the gates of the compression circuit in reverse order with an added difference of having controlled X_{-1} gates instead of controlled X_{+1} gates. The truth table of the 2-3-1 compression scheme is shown in Table III. By substituting values of the qubits A, B, C in the circuit and performing higher level qudit operations, it can be verified that the compression yields corresponding A', B' (qutrits) and C' (ancilla in $|0\rangle$ state) entries from the truth table and decompression yields back original values of A, B, and C.

TABLE III
TRUTH TABLE FOR 2-3-1 COMPRESSION SCHEME [48].

A	B	C	A'	B'	C'
0	0	0	0	0	0
0	0	1	2	2	0
0	1	0	0	1	0
0	1	1	0	2	0
1	0	0	1	0	0
1	0	1	2	1	0
1	1	0	1	1	0
1	1	1	1	2	0

*f) **Approximate PQC-based QRAM**:* A trainable PQC-based QRAM [49] similar to EQGAN QRAM is proposed that is able to store data in the quantum Hilbert space by training the PQC. Compared to the EQGAN, the approximate PQC-based QRAM does not store data in superposition but one-by-one in sequential order and is able to store more complex datasets such as image datasets like the UCI digits dataset. The approximate PQC-based QRAM is also used for the pure storage of binary data. The detailed PQC of the approximate QRAM is shown in Fig. 9. It consists of an embedding scheme such as angle, amplitude, or basis embedding to load classical data, followed by 3 sets of circular layers and strongly entangling layers respectively. It has been noted that loading images from QRAM and sending them to a QNN yields faster convergence of classification (by 6^{th} epoch) as compared to loading images without QRAM (around 15^{th} epoch), and for pure storage the QRAM is able to store 4-bit binary data without any errors.

C. Where is a QRAM used?

A QRAM that is able to store and load data in superposition is very helpful for certain classes of quantum algorithms.

- *Database search:* Grover's algorithm [16] and its more generalized version Quantum Amplitude Amplification and Estimation (QAE) [17] have been proposed to perform a database search for an element out of n elements with complexity O($\sqrt{n}$). They take data in superposition as input and perform the amplification operation O($\sqrt{n}$) times prior to performing estimation with a reduced number of measurements.
- *Element distinctness:* Given a set of n elements, the element distinctness problem asks whether all n elements in the set are distinct. Classically it takes O(n log(n)) time

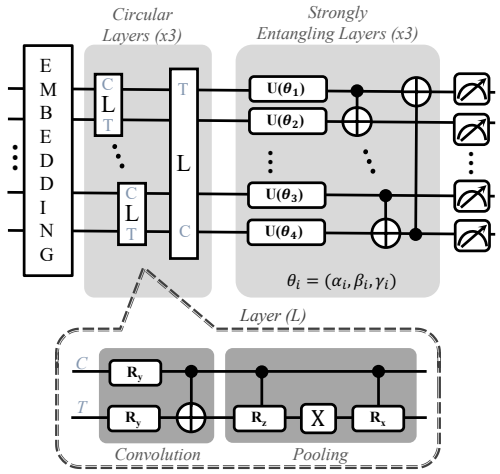

Fig. 9. PQC structure of approximate PQC-based QRAM. In the given context, 'C' symbolizes the control qubit, while 'T' stands for the target qubit.

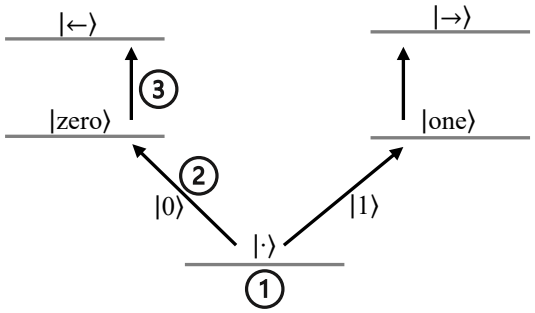

Fig. 10. Energy levels of trapped atom-based qutrit switches in bucket-brigade QRAM. ① Initialized state of qutrit. ② First incoming photon getting absorbed and changing the state of the qutrit and routing it either in |zero⟩ or |one⟩ direction based on the state of the qubit encoded in the photon. ③ Subsequent photons getting absorbed to $|\leftarrow\rangle$ or $|\rightarrow\rangle$ and remitted to the next qutrit based on the state of the previous qubit.

while quantum algorithms such as [19] solve it in $O(n^{\frac{2}{3}})$ time.

- *Collision detection:* Collision detection is an important problem in cryptography. Given a collision function H, the collision detection problem asks to find two distinct inputs x and y such that $H(x) = H(y)$. Quantum versions of the collision detection problem such as [18] report $O(n^{\frac{1}{3}})$ runtime where n denotes the cardinality of the domain of collision function.
- *NAND tree evaluation:* In this problem, a Boolean expression is solved using a tree of NAND gates. For an input of size n, quantum algorithms such as [50] propose a runtime of $O(\sqrt{n})$.
- *Quantum forking:* In classical operating systems, forking is the process of creating a child process from a parent process which is a copy of it, while retaining the parent process. Quantum forking is a similar idea, where the QRAM output superposition state is forked on ancilla qubits and then both the original state and forked state are multiplied with the same or different unknown unitaries. The new states then undergo a swap-test procedure to verify if the unitaries applied are the same or different [44], [51].
- *Storage of classical data:* As mentioned previously, works such as [46], [49] use a PQC-based QRAM circuit to store classical data such as data from a normal distribution, images, and binary data into quantum Hilbert space by training the PQC like a machine learning model.

IV. Practicality of QRAM

Follow-up paper [37] on [36] provide possible physical implementations of bucket-brigade QRAM and fanout QRAM. We first explain these implementations followed by implementation details on the FF-QRAM, qudits-based storage, and trainable PQC-based QRAM. We also show a detailed tabular comparison of different QRAMs in Table IV.

*a) **Bucket-Brigade QRAM implementation**:* For physically implementing the bucket-brigade QRAM, the authors in [37] incorporate (i) address qubits in the input register as photons that can be sent sequentially and (ii) the qutrits as trapped atoms inside cavities. The qubits encoded in the photons traverse the cavity by encountering the trapped atom-based qutrits. The three states of the qutrits are realized as three different energy levels, with the $|\cdot\rangle$ being the lowest energy state, with two higher energy levels |zero⟩ that is coupled with the left spatial path i.e., to the left qutrit along the bifurcation graph and |one⟩ that is coupled to the right spatial path to the right qutrit. This coupling is represented using further higher energy states $|\leftarrow\rangle$ (for left spatial path) and $|\rightarrow\rangle$ (for right spatial path). We show the energy diagram of the qutrit switches in Fig. 10.

Initially, all the qutrits are initialized to the lowest energy state $|\cdot\rangle$. When the first photon traverses through the cavity and reaches the root node switch, it gets absorbed into the higher energy state of the qutrit, either |zero⟩ or |one⟩ thereby changing the state of the qutrit depending on the quantum state encoded in the photon. This process is often referred to as the Raman transition, where a photon is scattered by a molecule, resulting in a change in the energy of the photon and the vibrational state of the molecule [52]. This is achieved with the help of strong laser fields [53] that help in changing the state of the qutrit from $|\cdot\rangle$ to |zero⟩ if the photon state is $|0\rangle$ and from $|\cdot\rangle$ to |one⟩ if the photon state is $|1\rangle$. After this, when the second photon arrives at the root node switch it once again gets absorbed and undergoes a Raman transition, but this time either from |zero⟩ to $|\leftarrow\rangle$ or |one⟩ to $|\rightarrow\rangle$ and will be remitted to the qutrit along the correct spatial path based on the state of the qutrit (|zero⟩: left path, |one⟩: right path). In this way, all the photons of the input register set the qutrit switches one by one until a path from the root node switch to the desired memory cell is created. The output register then either loads contents from the memory cell or stores new data in it via the created path of qutrit switches. Once the load/store operation is complete, all the qutrits starting from the last node to the root node undergo a final Raman transition sequentially to go back to $|\cdot\rangle$ state.

A recent work [43] proposed a quantum circuit-based implementation of the bucket-brigade QRAM. For n address

TABLE IV
TABLE OF COMPARISON BETWEEN DIFFERENT QRAM TECHNOLOGIES.

Feature	**Bucket-Brigade QRAM [36]**	**Fanout QRAM [37]**	**Flip-Flop QRAM [44]**	**Qudits-based memory [48]**	**Approximate PQC-based [49] & EQGAN QRAM [46]**
Structure	Bifurcation graph	Bifurcation graph	Quantum circuit	Higher states	Parametric Quantum Circuit
Circuit width ($n = \#address\ lines$)	O(2^n)	O(2^n)	O(n)	Dependent on d (# qudit states)	O(n)
Circuit depth ($n = \#address\ lines$)	O(2^n)	O(2^n)	O(2^n)	Dependent on d (# qudit state)	O(1)
Unique qualities	Qubits are routed in a sequential fashion	Qubits controlling exponential quantum switches	Quantum circuit-based	Reduces requirements of ancilla qubits to 0	Trainable like a machine learning model
Implementation technology	Photons, trapped atoms	Photons, microwave cavities	Superconducting qubits, trapped ion qubits	Superconducting qudits, trapped ion qudits, OAM photonic qudits	Superconducting qubits, trapped ion qubits
Drawbacks	Exponential circuit width and depth	Exponential circuit width and depth, susceptible to decoherence	Exponential circuit depth	Unstable higher states	Performance degradation under noise (approx. QRAM), store only simple dataset (EQGAN)

lines, the quantum circuit requires O(n) qubits for address, O(2^n) ancilla qubits for incorporating the quantum switches, O(2^n) qubits for memory cells and one qubit for readout of the memory cell. We show an implementation of quantum circuit-based bucket-brigade QRAM for two address lines and four memory cells in Fig. 4 where the memory cell in address $|01\rangle$ is being accessed. First, $|a_1\rangle = |0\rangle$ changes the state of the first ancilla qubit which then changes the state of the next ancilla qubit. Based on the output, the path is then routed to the left child switch, where $|a_0\rangle = |1\rangle$ is used to switch the right ancilla qubit to $|1\rangle$ state. Finally, a set of Toffoli gates with ancilla qubit as one control and the memory cell qubit as another control are used to perform readout. Depending on the address state, only the Toffoli gate controlled by its corresponding memory cell will be triggered. In this case, the Toffoli gate corresponding memory cell $|m_{01}\rangle$ is triggered to perform a readout operation on the readout qubit.

*b) **Fanout QRAM implementation**:* Two implementations, namely optical implementation and phase gate implementation have been proposed for the fanout QRAM [37]. Understanding the implementations in the original paper may be challenging, as it assumes knowledge of optical and cavity-based quantum systems.

In the phase gate implementation, (i) the address qubits in the input register are photons, and (ii) the quantum switches are also photonic qubits trapped inside microwave cavities. Overall, for n address qubits there are O(2^n) microwave cavities with each cavity containing a photonic qubit. As mentioned earlier, the k^{th} index qubit fans out and controls 2^k quantum switches. This is achieved using conditional phase shifters. The MSB address qubit will only polarize the root node photon inside the microwave cavity via the conditional phase shifter attached to it. The next address qubit will polarize two child photons via a conditional phase shifter. This continues until all the photons inside the microwave cavity are polarized. As a result of this, a resonant path will be created from the root cavity to the desired memory cell. Each memory cell consists of two superconducting qubits, such that one is for storing information, and one is for extracting information. The contents of the memory cell are then transferred back to the output register using a SWAP gate via an outgoing photon from the memory cell with the help of the extraction qubit.

Next, we discuss the optical implementation. Here, (i) the address qubits are atoms trapped in a magneto-optical trap and (ii) the quantum switches are photonic qubits that hit the trapped atomic address qubits one by one. When the first quantum switch photon hits the first address qubit inside the trapped atom, the trapped atom acts as a controller for changing the polarization state of the photon. This photon then passes through a polarization beam splitter and a half-wave plate to transfer this information to another spatial degree of freedom and create two spatial modes. The two spatial modes will be two photonic quantum switches for the next address qubit. Once again, each spatial mode will transfer the state of the address qubit via a change of polarization and create two new modes. This continues until 2^n spatial modes are created, one for each memory cell. Out of these, only one spatial mode will be active depending on the address, and the contents of the desired memory cell corresponding to the active spatial mode are swapped out with the contents of the output register using a SWAP gate.

*c) **EQGAN QRAM implementation**:* Since EQGAN QRAM is a quantum circuit-based QRAM, it can be implemented on superconducting and trapped ion qubits. The authors in [46] implement the EQGAN QRAM on 5 qubits of Google's Sycamore superconducting processor such that the readout qubit (top qubit in Fig. 7) is the center physical qubit and the rest of the qubits are physically coupled with the readout qubit in the shape of a (+) sign on a grid of qubits.

*d) **Qudit implementation**:* Qudits are implementable on physical quantum systems that have an infinite spectrum of states, like superconducting qubits (magnetic flux spectrum) [54], trapped ion qubits (energy band spectrum) [55], and Orbital Angular Momentum (OAM spectrum) based photonic

qubits [56]. For example, in bucket-brigade QRAM, the qutrit switches are implemented using trapped atoms in a cavity, with the $|\cdot\rangle$ state at a lower energy level and $|\text{zero}\rangle$ and $|\text{one}\rangle$ states at a higher energy level.

*e) **Approximate PQC-based QRAM & Flip-Flop QRAM implementations**:* Similar to the EQGAN QRAM, since both of these QRAMs are quantum-circuit based they can be implemented on superconducting and trapped ion qubits. With the quantum circuit known, the QRAM architectures can be replicated on known quantum computing platforms such as Qiskit [57] from IBM, Pennylane [58] from Xanadu, IonQ [59] and many more. The users can replicate the quantum circuit and send it either for simulation on a noiseless/noisy simulator (better for approximate PQC-based QRAM [49] as it is iterative), or run it on actual quantum hardware (better for FF-QRAM [44] as it is non-iterative).

V. Challenges and Future Direction

In this section, we examine the current limitations and future directions of various QRAM architectures and the challenges. Some of the common challenges include:

- *Scalability:* Scalability is a major challenge in QRAM designs due to constraints in qubit interactions, quantum memory, and coherence. Increasing memory elements in bucket-brigade, fanout, and FF-QRAMs leads to exponential growth in circuit width and depth. Thus, scalability remains a significant hurdle for large-scale QRAM implementations.
- *Noise resilience:* Noise resilience is a crucial challenge in QRAM architectures, as quantum systems are sensitive to environmental noise. In various QRAM types, increasing memory elements results in higher circuit depth and qubit count, making the system more vulnerable to noise. Bucket brigade QRAM is comparatively more noise-resilient than fanout QRAM [41], while FF-QRAM is susceptible to noise with increasing address lines. PQC-based QRAMs have constant circuit depth but are still prone to noise-related errors, affecting performance on real hardware versus simulation.
- *No-cloning theorem:* The no-cloning theorem [38], [60], a fundamental quantum mechanics principle, prohibits exact copying of unknown quantum states and poses challenges for various QRAM designs. In bucket brigade and fanout QRAM, the theorem limits duplication of quantum states during memory readout. Solutions like CNOT or SWAP gates are available, but the no-cloning theorem still complicates error correction and redundancy schemes in most QRAM designs, presenting a significant challenge [23].
- *Instability of qudits:* Qudit instability primarily affects qudit-based quantum memory, where qudits are quantum systems with $d > 2$ levels. While qudit-based memory can store more information than qubit-based systems, higher qudit states are unstable and prone to errors [61], [62]. For example, the energy gap between higher states in superconducting qubits is less [1]. This issue is not directly relevant to qubit-based QRAM designs, but incorporating qudits for increased storage would introduce similar challenges related to qudit instability.
- *Limited applicability:* Some QRAM architectures face limitations due to their novelty, experimental difficulties, or specific focus. For example, FF-QRAM has limited applicability beyond quantum forking due to its targeted design. Similarly, qudits face challenges arising from limited research and increased complexity compared to qubits. Addressing these challenges is essential for advancing broader applications in quantum computing.

While current QRAM designs face the above challenges, ongoing research strives to overcome them. Several recent works have made significant progress in the implementation of bucket brigade QRAM. In one study [63], researchers construct a circuit implementation of the aforementioned QRAM, demonstrating that when used with classical data, it can quickly and repeatedly prepare arbitrary quantum states once the data is already present in memory. Another work [64] discusses the parallelization of queries in a bucket-brigade QRAM, showing that the parallelization method is compatible with surface code quantum error correction. In theory, fault-tolerant bucket-brigade QRAM queries can be performed at speeds comparable to classical RAM. A separate article [43] addresses the robustness of bucket brigade QRAM, revealing that when quantum error correction is applied to the bucket brigade QRAM circuit, the circuit loses its advantage of having a small number of active gates, as error correction operates on all of its components.

The precise evaluation of the hardware expenditure associated with QRAM designs, especially in the realm of fault-tolerant systems, may constitute a significant subject of forthcoming research [23]. It is reasonable to anticipate that when compared to conventional surface code implementations [65], the hardware expenses and intricacy will be substantially reduced owing to the noise resilience inherent in bucket-brigade QRAM and the implementation of low-overhead fault tolerance techniques utilizing qubits. While there have been notable advancements in hardware efficiency, it remains a challenge to develop a QRAM capable of addressing millions or billions of individual memory elements in the near future. Exploring applications where smaller QRAMs can provide value and conducting tailored resource estimations for these use cases could be key for future progress and development.

VI. Conclusion

Quantum Random Access Memory (QRAM) serves as a specialized form of memory that enables direct access and manipulation of quantum states, thereby facilitating expedited and efficient data retrieval and storage within quantum systems. Unlike conventional RAM structures that store information in classical bits, which are incompatible with quantum systems, QRAM operates on the principles of quantum computing. This enables QRAM to store and manipulate quantum data effectively, resulting in considerable acceleration of known

quantum algorithms. This review provides a thorough assessment of QRAM, emphasizing its importance and practicality within the context of contemporary quantum computing. We explain the fundamentals of quantum computing and conventional RAM, before delving into the foundations of QRAM. Five notable types of QRAM designs are outlined, including bucket-brigade QRAM, fanout QRAM, flip-flop QRAM, qudits-based quantum memory, and approximate PQC-based QRAM. By comparing these diverse architectural approaches and carefully analyzing their implementation, the feasibility of QRAM is thoroughly explored. The analysis concludes by discussing the primary challenges associated with QRAM development and future directions.

REFERENCES

[1] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied physics reviews*, vol. 6, no. 2, p. 021318, 2019.

[2] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, p. 021314, 2019.

[3] S. Slussarenko and G. J. Pryde, "Photonic quantum information processing: A concise review," *Applied Physics Reviews*, vol. 6, no. 4, p. 041303, 2019.

[4] Y. Arakawa and M. J. Holmes, "Progress in quantum-dot single photon sources for quantum information technologies: A broad spectrum overview," *Applied Physics Reviews*, vol. 7, no. 2, p. 021309, 2020.

[5] S. Pezzagna and J. Meijer, "Quantum computer based on color centers in diamond," *Applied Physics Reviews*, vol. 8, no. 1, p. 011308, 2021.

[6] M. Schuld, I. Sinayskiy, and F. Petruccione, "An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, no. 2, pp. 172–185, 2015.

[7] D. Herman, C. Googin, X. Liu, A. Galda, I. Safro, Y. Sun, M. Pistoia, and Y. Alexeev, "A survey of quantum computing for finance," *arXiv preprint arXiv:2201.02773*, 2022.

[8] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. Sawaya *et al.*, "Quantum chemistry in the age of quantum computing," *Chemical reviews*, vol. 119, no. 19, pp. 10 856–10 915, 2019.

[9] P. Wallden and E. Kashefi, "Cyber security in the quantum era," *Communications of the ACM*, vol. 62, no. 4, pp. 120–120, 2019.

[10] F. Bova, A. Goldfarb, and R. G. Melko, "Commercial applications of quantum computing," *EPJ quantum technology*, vol. 8, no. 1, p. 2, 2021.

[11] S. Zhou, T. Loke, J. A. Izaac, and J. Wang, "Quantum fourier transform in computational basis," *Quantum Information Processing*, vol. 16, pp. 1–19, 2017.

[12] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[13] R. Schützhold, "Pattern recognition on a quantum computer," *Physical Review A*, vol. 67, no. 6, p. 062311, 2003.

[14] G. Schaller and R. Schützhold, "Quantum algorithm for optical-template recognition with noise filtering," *Physical Review A*, vol. 74, no. 1, p. 012303, 2006.

[15] C. A. Trugenberger, "Probabilistic quantum memories," *Physical Review Letters*, vol. 87, no. 6, p. 067901, 2001.

[16] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[17] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," *Contemporary Mathematics*, vol. 305, pp. 53–74, 2002.

[18] G. Brassard, P. Høyer, and A. Tapp, "Quantum cryptanalysis of hash and claw-free functions," *ACM Sigact News*, vol. 28, no. 2, pp. 14–19, 1997.

[19] A. Ambainis, "Quantum walk algorithm for element distinctness," *SIAM Journal on Computing*, vol. 37, no. 1, pp. 210–239, 2007.

[20] A. M. Childs, A. W. Harrow, and P. Wocjan, "Weak fourier-schur sampling, the hidden subgroup problem, and the quantum collision problem," in *STACS 2007: 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007. Proceedings 24*. Springer, 2007, pp. 598–609.

[21] M. Schuld and F. Petruccione, *Supervised learning with quantum computers*. Springer, 2018, vol. 17.

[22] O. Di Matteo, V. Gheorghiu, and M. Mosca, "Fault-tolerant resource estimation of quantum random-access memories," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–13, 2020.

[23] C. T. Hann, "Practicality of quantum random access memory," Ph.D. dissertation, Yale University, 2021.

[24] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[25] E. G. Rieffel and W. H. Polak, *Quantum computing: A gentle introduction*. MIT Press, 2011.

[26] A. Y. Kitaev, A. Shen, M. N. Vyalyi, and M. N. Vyalyi, *Classical and quantum computation*. American Mathematical Soc., 2002, no. 47.

[27] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels," *Physical review letters*, vol. 70, no. 13, p. 1895, 1993.

[28] S. Lloyd, "Universal quantum simulators," *Science*, vol. 273, no. 5278, pp. 1073–1078, 1996.

[29] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[30] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. H. Booth *et al.*, "The variational quantum eigensolver: a review of methods and best practices," *Physics Reports*, vol. 986, pp. 1–128, 2022.

[31] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," *Physical review letters*, vol. 113, no. 13, p. 130503, 2014.

[32] T. Hey, "Richard feynman and computation," *Contemporary Physics*, vol. 40, no. 4, pp. 257–265, 1999.

[33] R. C. Jaeger, T. N. Blalock, and B. J. Blalock, *Microelectronic circuit design*. McGraw-Hill New York, 1997.

[34] D. A. Patterson and J. L. Hennessy, *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann, 2016.

[35] W. Stallings, *Computer organization and architecture: designing for performance*. Pearson Education India, 2003.

[36] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum random access memory," *Physical review letters*, vol. 100, no. 16, p. 160501, 2008.

[37] ——, "Architectures for a quantum random access memory," *Physical Review A*, vol. 78, no. 5, p. 052310, 2008.

[38] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, pp. 802–803, 1982.

[39] M. Schlosshauer, "Quantum decoherence," *Physics Reports*, vol. 831, pp. 1–57, 2019.

[40] J. Von Neumann, *Mathematical foundations of quantum mechanics: New edition*. Princeton university press, 2018, vol. 53.

[41] C. T. Hann, G. Lee, S. Girvin, and L. Jiang, "Resilience of quantum random access memory to generic noise," *PRX Quantum*, vol. 2, no. 2, p. 020311, 2021.

[42] P. Horowitz, W. Hill, and I. Robinson, *The art of electronics*. Cambridge university press Cambridge, 1989, vol. 2.

[43] S. Arunachalam, V. Gheorghiu, T. Jochym-O'Connor, M. Mosca, and P. V. Srinivasan, "On the robustness of bucket brigade quantum ram," *New Journal of Physics*, vol. 17, no. 12, p. 123010, 2015.

[44] D. K. Park, F. Petruccione, and J.-K. K. Rhee, "Circuit-based quantum random access memory for classical data," *Scientific reports*, vol. 9, no. 1, p. 3949, 2019.

[45] T. M. De Veras, I. C. De Araujo, D. K. Park, and A. J. Da Silva, "Circuit-based quantum random access memory for classical data with continuous amplitudes," *IEEE Transactions on Computers*, vol. 70, no. 12, pp. 2125–2135, 2020.

[46] M. Y. Niu, A. Zlokapa, M. Broughton, S. Boixo, M. Mohseni, V. Smelyanskyi, and H. Neven, "Entangling quantum generative adversarial networks," *Physical Review Letters*, vol. 128, no. 22, p. 220505, 2022.

[47] P. Gokhale, J. M. Baker, C. Duckering, N. C. Brown, K. R. Brown, and F. T. Chong, "Asymptotic improvements to quantum circuits via qutrits,"

in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 554–566.

[48] J. M. Baker, C. Duckering, and F. T. Chong, "Efficient quantum circuit decompositions via intermediate qudits," in *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2020, pp. 303–308.

[49] K. Phalak, J. Li, and S. Ghosh, "Approximate quantum random access memory architectures," *arXiv preprint arXiv:2210.14804*, 2022.

[50] A. M. Childs, B. W. Reichardt, R. Spalek, and S. Zhang, "Every nand formula of size n can be evaluated in time nˆ{1/2+ o (1)} on a quantum computer," *arXiv preprint quant-ph/0703015*, 2007.

[51] D. K. Park, I. Sinayskiy, M. Fingerhuth, F. Petruccione, and J.-K. K. Rhee, "Parallel quantum trajectories via forking for sampling without redundancy," *New Journal of Physics*, vol. 21, no. 8, p. 083024, 2019.

[52] M. Feng, "Quantum computing with trapped ions in an optical cavity via raman transition," *Physical Review A*, vol. 66, no. 5, p. 054303, 2002.

[53] G. Moy, J. Hope, and C. Savage, "Atom laser based on raman transitions," *Physical Review A*, vol. 55, no. 5, p. 3631, 1997.

[54] T. Liu, Q.-P. Su, J.-H. Yang, Y. Zhang, S.-J. Xiong, J.-M. Liu, and C.-P. Yang, "Transferring arbitrary d-dimensional quantum states of a superconducting transmon qudit in circuit qed," *Scientific reports*, vol. 7, no. 1, p. 7039, 2017.

[55] P. J. Low, B. M. White, A. A. Cox, M. L. Day, and C. Senko, "Practical trapped-ion protocols for universal qudit-based quantum computing," *Physical Review Research*, vol. 2, no. 3, p. 033128, 2020.

[56] N. Bent, H. Qassim, A. Tahir, D. Sych, G. Leuchs, L. L. Sánchez-Soto, E. Karimi, and R. Boyd, "Experimental realization of quantum tomography of photonic qudits via symmetric informationally complete positive operator-valued measures," *Physical Review X*, vol. 5, no. 4, p. 041006, 2015.

[57] "IBM Quantum," 2023. [Online]. Available: https://quantum-computing.ibm.com/

[58] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018.

[59] "IonQ," 2023. [Online]. Available: https://ionq.com/

[60] D. Dieks, "Communication by epr devices," *Physics Letters A*, vol. 92, no. 6, pp. 271–272, 1982.

[61] M. Grassl, L. Kong, Z. Wei, Z.-Q. Yin, and B. Zeng, "Quantum error-correcting codes for qudit amplitude damping," *IEEE Transactions on Information Theory*, vol. 64, no. 6, pp. 4674–4685, 2018.

[62] B. P. Lanyon, T. J. Weinhold, N. K. Langford, J. L. O'Brien, K. J. Resch, A. Gilchrist, and A. White, "Manipulating biphotonic qutrits," *Physical review letters*, vol. 100, no. 6, p. 060504, 2008.

[63] P. A. M. Casares, "Circuit implementation of bucket brigade qram for quantum state preparation," *arXiv preprint arXiv:2006.11761*, 2020.

[64] A. Paler, O. Oumarou, and R. Basmadjian, "Parallelizing the queries in a bucket-brigade quantum random access memory," *Physical Review A*, vol. 102, no. 3, p. 032608, 2020.

[65] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002.