Quantum Information and Computing Final Project
Quantum Random Access Memory (QRAM): Its Architectures and Applications

黃紹凱（B12202004）[1]、陳昱綸（B12901107）[2]、劉育成（B11901182）[3]

[1]) Department of Physics, National Taiwan University
[2], [3]) Department of Electrical Engineering, National Taiwan University

Here we present an overview on the various architectures, implementations, and research progress in the quantum random access memory.

1 Introduction to Quantum RAM (QRAM)

1.1 Background

Classical random-access memory (RAM) lets an n-bit address select *any* one of 2^n stored words in $O(1)$ time. Figure 1 shows the textbook organization adopted by static RAM (SRAM) and dynamic RAM (DRAM) chips alike [8, 9].

Cell array. At the physical level every data bit lives at the intersection of a *word-line* (horizontal) and a pair of complementary *bit-lines* (vertical). In SRAM each cell is a bistable six-transistor latch; in DRAM it is a single transistor plus a tiny capacitor. Cells share their bit-lines column-wise, so only one entire row is active at a time.

Row decoder. The n-bit address is split into a row field and (in DRAM) a column field. A binary tree of pass-transistors decodes the row bits and asserts exactly one word-line. Activating the word-line connects every cell in that row to its two bit-lines, placing either a small differential current (SRAM) or a tiny charge redistribution (DRAM) onto the columns.

Sense amplifiers and column logic. Differential sense amplifiers at the bottom of the columns detect which bit-line pair is slightly higher in voltage and latch the decision within a few nanoseconds. Optional column decoders or multiplexers then select the w-bit word that forms the data output. Because the bit-lines are long metal buses with $\mathcal{O}(p\mathrm{F})$ capacitance, most of the access latency and power is spent charging and discharging them.

Timing model. From the programmer's vantage point the entire path

$$\text{address} \longrightarrow \text{row activate} \longrightarrow \text{sense / restore} \longrightarrow \text{data out}$$

takes a fixed time $t_{\mathrm{RC}} + t_{\mathrm{sense}}$, independent of the numeric value of the address, hence the term "random access". Modern DDR-x DRAM pipelines this sequence so that a new address can be issued every cycle even though an individual access still spans multiple cycles.

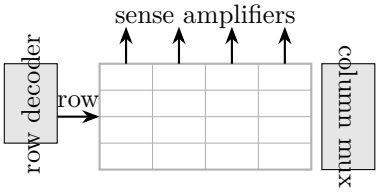

Figure 1: Simplified organization of a classical $2^n \times w$ RAM array. One word-line is activated by the row decoder; differential sense amplifiers on the bit-lines produce the data word.

This constant-time, address-independent abstraction is precisely what a quantum random-access memory aims to preserve, but now with the additional requirement that the address register may be in an arbitrary superposition—and that the memory must entangle the *correct* data word with each branch of that superposition.Given an input state $\sum_k \alpha_k |k\rangle_Q$, an ideal QRAM performs the isometry

$$\sum_k \alpha_k |k\rangle_Q |0\rangle_A \longrightarrow \sum_k \alpha_k |k\rangle_Q |f_k\rangle_A , \tag{1}$$

so one query loads *all* requested records in parallel [5]. That simple promise—address superposition with data-dependent entanglement—makes QRAM a foundational primitive for scalable quantum information processing.

1.2 Motivation

Quantum random-access memory is attractive because it removes the *input bottleneck* that plagues many otherwise promising quantum algorithms. Classically, loading an N-item data set requires $\Theta(N)$ time and energy, so even an $O(\sqrt{N})$ quantum speed-up disappears if the oracle is implemented by sequential I/O on a control computer. A coherent QRAM query, by contrast, transfers *all* N records into superposition using $\mathrm{polylog}\, N$ hardware depth [5, 16]. That feature is the linchpin of three research directions.

Linear-systems and simulation algorithms. The Harrow—Hassidim—Lloyd (HHL) solver prepares the vector $|x\rangle = A^{-1}|b\rangle$ in time $\widetilde{O}(\kappa^2 \log N)$ once the $|b\rangle$ register is available in amplitude encoding [7]. Without QRAM, loading $|b\rangle$ already costs $\Theta(N)$; with QRAM the loader is asymptotically hidden inside the polylog overhead.

Quantum machine learning. Most quantum supervised- and unsupervised-learning proposals—quantum support-vector machines [17], quantum PCA [12], quantum recommendation systems [11] —begin by mapping a classical feature vector $\mathbf{v} \in \mathbb{R}^N$ to the quantum state $|v\rangle = \frac{1}{\|\mathbf{v}\|}\sum_j v_j |j\rangle$. Efficient state preparation therefore *defines* the usefulness of the algorithm; QRAM is the method most often assumed in the complexity analyses [18].

Streaming and on-the-fly data. Emerging "quantum RAM-disk'' designs propose to couple a cryogenic classical memory die directly to a dilution-refrigerator quantum processor, so that experiment data, random seeds or stochastic oracle coefficients can be swapped in and out at run time [21].[1] In that scenario the loader is invoked many times with different content; its fault tolerance and energy per call become as critical as its asymptotic depth.

Taken together, these lines of work justify treating QRAM as a core *memory hierarchy layer* for future quantum accelerators rather than as a niche gadget for a handful of algorithms.

1.3 Representative Application Scenarios

Grover's Algorithm. The classic example is Grover's unstructured search algorithm [6]. QRAM supplies the oracle

$$|k\rangle|0\rangle \mapsto |k\rangle|f_k\rangle,$$

after which a conditional phase flip marks the solution. Because the oracle is reversible the entire amplitude-amplification loop preserves coherence, yielding the familiar $O(\sqrt{N})$ query complexity.

Quantum Minimal Search. A closely related primitive is *quantum minimum search* (QMS). Dürr and Høyer showed that Grover iterations plus an $O(\log N)$ classical update register find the global minimum of a list in $\widetilde{O}(\sqrt{N})$ queries [3]. Recent refinements replace the classical register by an *incremental QRAM* that updates only those cells whose value falls below the running minimum, cutting circuit depth by a constant factor [15].

Quantum KNN. In quantum k-nearest-neighbor (Q-KNN) classification, each training vector $\mathbf{v}^{(i)}$ is stored in one QRAM cell. A single query prepares $\sum_i |i\rangle \left|v^{(i)}\right\rangle |x\rangle$, where $|x\rangle$ encodes the test point. A swap test then estimates all Euclidean distances in parallel, so the decision takes $O(\sqrt{N})$ time instead of $O(N)$ [20]; follow-up work improves success probability by combining QRAM with amplitude-estimation subroutines [22].

Quantum Amplitude Estimation. Beyond search and classification the *quantum amplitude estimation* (QAE) family uses QRAM-backed data oracles to accelerate Monte-Carlo pricing of financial derivatives [10] and to compute risk measures such as *Value-at-Risk* or *Expected Shortfall*. Here the oracle prepares a payoff distribution while QAE reduces the sampling error quadratically, giving an end-to-end speed-up provided the QRAM call costs less than $O(\sqrt{N})$ classical samples—which holds as long as the address register fits on available hardware.

Variational Quantum Algorithms. A final, more speculative, direction embeds QRAM inside variational quantum algorithms (VQAs): the loader initialises a parameterised state $|\psi(\theta)\rangle$ with data-dependent angles; a shallow variational ansatz then refines the state before measurement [4]. Early numerical experiments suggest that QRAM-initialised VQAs can converge in fewer optimisation steps than randomly initialised ones, albeit at the price of higher circuit width.

The shared lesson across all these scenarios is that *QRAM calls are rarely standalone.* They appear inside larger algorithmic loops whose depth and query count determine the effective noise tolerance (ğ 2). Understanding that system-level context is therefore essential when evaluating any proposed QRAM implementation.

[1] Such proposals are at the proof-of-concept level but exemplify an architectural motivation distinct from purely algorithmic speed-ups.

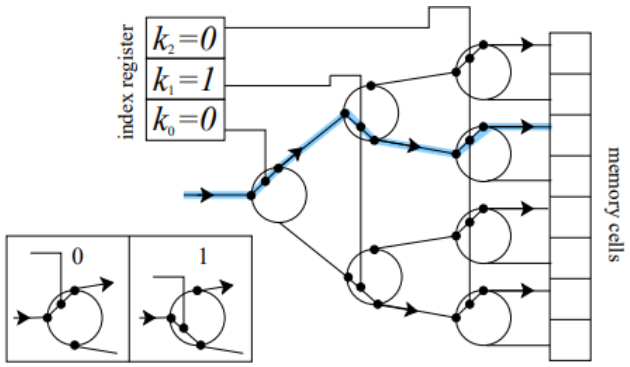

Figure 2: Demonstration of the classical fanout scheme.

2 Comparison of QRAM Architectures

2.1 Overview

With the advancement of hardware and qubit technologies, the possibility of implementing quantum algorithms, such as Grover's algorithm, on a quantum computer becomes more and more within reach. The is section introduces the relevant technologies for physical implementation in further detail, and discusses a variety of potential QRAM architectures. This section describes several proposed QRAM architectures and evaluates them based on structure, complexity, platform, and practical considerations. Also, since it is easier to compare and demonstrate the quantum advantage (albeit only intuitively) of QRAM, we will also elaborate on their respective classical counterparts.

Table 1: Comparison of QRAM Architectures

Architecture	Circuit Width	Depth	Complexity	Implementation	Advantage
Bucket-Brigade	Bifurcation graph	$O(2^n)$	$O(2^n)$	Photonic	
Fanout QRAM	Bifurcation graph	$O(2^n)$	$O(2^n)$	Photonic	Simple control
Flip-Flop QRAM	Quantum circuit	$O(n)$	$O(2^n)$	Superconducting	Fewer ancilla
Qudit-based	Multi-level qudit	Depends on d	Depends on d	Ion/Photonic	Dense resources
PQC / EQGAN	Parametric QC	$O(n)$	$O(1)$	Universal QC	Trainable, QML-suitable

Next, we will provide a comprehensive review of various architectures in detail, including an introduction to their classical counterparts.

2.2 Fanout QRAM

In the *fanout QRAM* protocol, each bit of the address register controls multiple switches in a binary tree network, routing a quantum bus to the desired memory cell.

Classical Fanout RAM Scheme. In the classical fanout scheme, there is an *index register* that specifies the direction to follow to reach the memory cell we are interested in. Similar to the bucket-brigade architecture discussed above, the fanout protocol has a bifurcation graph circuit. Writing the index register in binary form, each bit of such register can be interpreted as an indicator of direction, so 2^n paths can be described.

We can realize this indexing procedure with a fanout RAM scheme, where the k-th index bit controls the value of 2^k (classical) bits in the k-th level of the tree. This uniquely specifies the path to the desired memory cell, but in the process it requires controlling all 2^n bits even though only n bits participate in addressing the memory cell. However, we can do a

The calssical fanout scheme is implemented in chips, by using electronic circuits and pairs of transistors to replace the binary tree and respective switches [**??**].

Quantization of the Fanout Scheme. The fanout control structure discussed above is easy to realize classically, but quantum implementations suffer from decoherence due to entanglement between address qubits and many control gates.

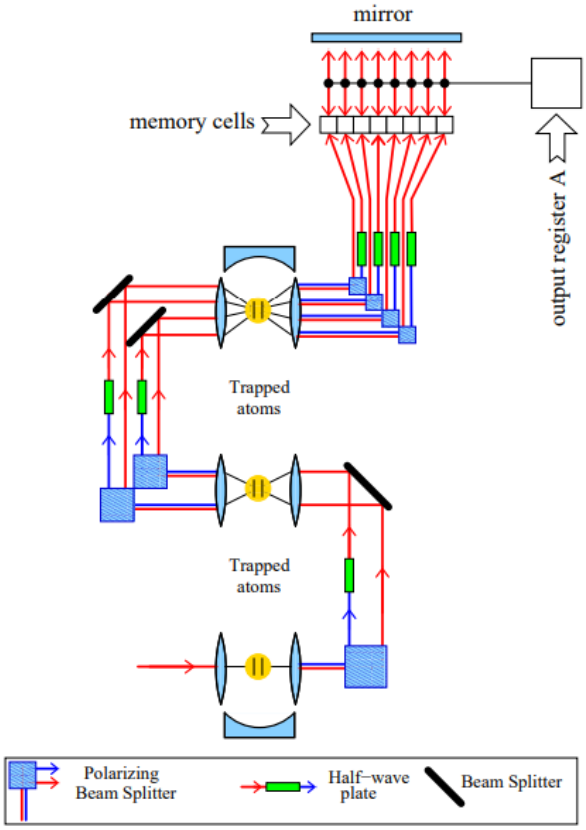

Figure 3: Optical implementation of fanout protocol.

Quantum versions require reversible operations and preservation of coherence through unitary transformations. Although scalable only to modest n, fanout QRAM is experimentally viable and foundational for understanding error propagation and decoherence in quantum memory systems.

Implementation. The quantum fanout precedure may be implemented with the following methods [**??**]:

1. Optical implementation.
2. Controlled phase gates implementation.

2.3 Bucket-Brigade QRAM

This is the first proposal for the structure of a QRAM. The architecture , unlike the traditional d-dimensional lattice of memory array used in classical RAM, uses a *bifurcation graph-based structure*. The structures is showcased in the diagrams [4], and also [5] later, taken from reference [**??**].

To understand the architecture, we need a knowledge of "qutrits", the generalization of qubits consisting of a two-level system $\{|0\rangle, |2\rangle\}$ to three-level systems. In a qutrit, apart from the two basis states $|0\rangle$ and $|1\rangle$, a third state usually denoted as $|\cdot\rangle$ is present, which indicates "sending the incoming signal on without changing its state". This comes in very handy in extending the protocol described in section 2.2 to be more efficient.

The QRAM Scheme. The bucket-brigade architecture is designed to address the inefficiencies of conventional fanout models. It utilizes a tree of three-state quantum systems (qutrits) that route address and data signals by sequential activation. Initially, all qutrits are in a passive state. As each bit of the address register propagates through the tree, qutrits transition to active states (0 or 1), encoding the routing path. The bus qubit follows this path to the target memory cell, then reverses its route to complete the transaction.

The advantage and thereof of the bucket-brigade QRAM scheme is discussed in the 2007 Giovannetti paper [**??**]. An advantage of the bucket-brigade method lies in its **robustness**. This is *

(q)RAM architecture reduces the number of switches that must be thrown during a RAM call, quantum or classical, from O(N1/d) to O(logN) , where N=2n is the number of memory slots in the RAM and d is the dimension of the lattice that, according to the authors, conventional RAM architectures use for memory retrieval.

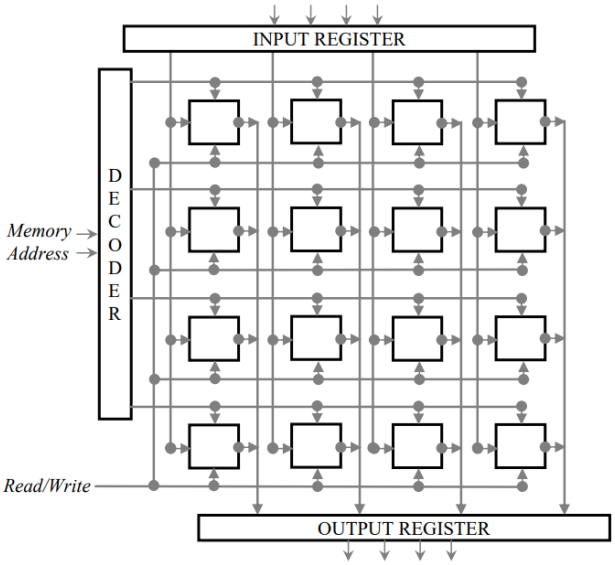

Figure 4: High dimensional lattice structure for a classical memory array.

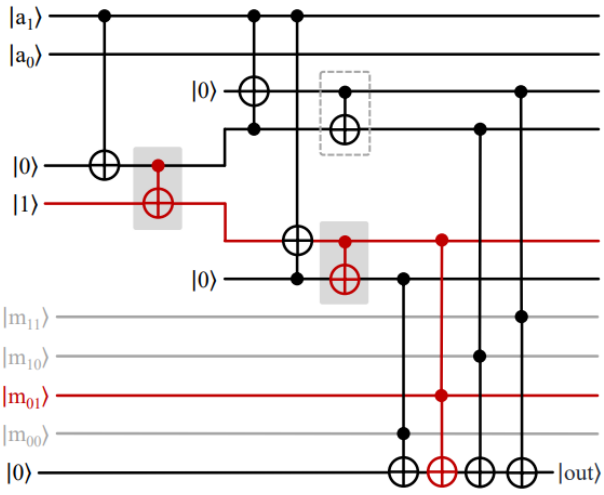

Figure 5: Bifurcation graph structure of the bucket-brigade protocol for QRAM implemented on a quantum circuit. In the diagram, data in the memory cell denoted $|m_{01}\rangle$ is being accessed via a sequence of quantum gates. Also, the red path represents the active route of the QRAM.

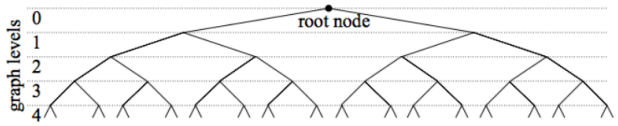

Figure 6: Bifurcation graph of "RAM addressing", which inspires the bucket-brigade protocol.

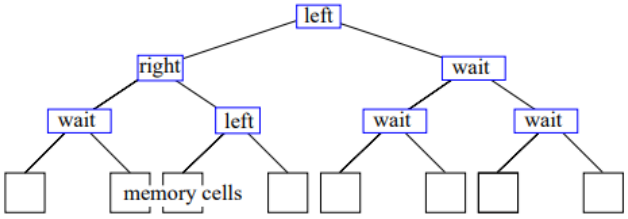

Figure 7: Illustration of the bucket-brigade protocol.

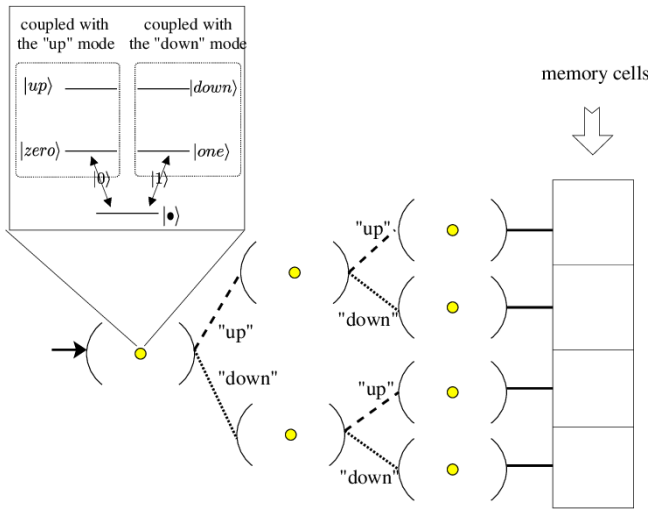

Figure 8: At each node of the above bifurcation tree, the three-state switch corresponding to a qutrit consists of an atom in a cavity.

Implementation. This method activates only $O(n)$ qutrits per memory call, compared to $O(2^n)$ in traditional models, significantly reducing decoherence risks and improving scalability. The architecture is suitable for photonic and atomic systems where controlled routing is feasible. Because the bucket-brigade qRAM operates by sequential coupling of qutrits, it takes $O(n^2)$ steps to retrieve one of the 2^n memories in a coherent manner.

Alternatively, it is mentioned that these "atoms in a cavity" need not be real atoms, but could be artificial atoms consisting of superconducting qubits.

2.4 Flip-Flop QRAM

Flip-Flop QRAM operates via dynamic switching between encoding and data-loading modes using a structured quantum circuit. Unlike fanout, it requires only $O(n)$ circuit width, significantly reducing the number of ancilla qubits. However, the circuit depth remains exponential due to sequential loading operations.

This architecture is primarily suitable for superconducting qubit systems, where rapid and precise switching can be implemented. Flip-flop QRAM offers a balance between resource usage and operational flexibility but demands complex timing control and synchronization.

2.5 Qudit-Based QRAM

Qudit-based QRAMs utilize multi-level systems (qudits) to encode address and data values compactly. This approach can dramatically reduce circuit width and depth depending on the dimensionality d of the qudits used. The primary advantage is dense information representation, making them suitable for high-capacity systems.

Fundamental Ideas. In section (**??**) we discussed qutrits and their application in , qudits make up a higher dimensional extension of traditional qubis and qutrits. In a traditional qubit there are two basis states: $|0\rangle$ and $|1\rangle$, and we tensor them to form a computational basis for a system. In a *qudit* system with d computational basis states, a quantum state becomes

$$|\psi\rangle = a_0 |0\rangle + a_1 |1\rangle + \cdots + a_{d-1} |d-1\rangle . \tag{2}$$

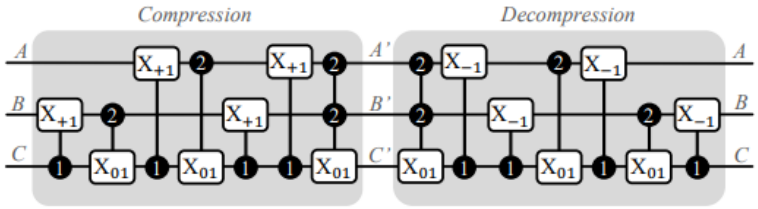

Figure 9: An example compression and decompression circuit, which compresses a system of three qubits A, B, C into a system of two qutrits A', B', and an ancilla C'.

Compared to quantum system composed of qubits, a *qudit* system with identical number of working bits can provide a larger state space to store and process information. Therefore, qudit and higher dimensional computing can provide reduction of the circuit complexity and simplification of the experimental setup [19], though the theory of qudits is not of main focus here.

Theory Behind Qudit QRAM. Qudits-based quantum memory has been proposed, where qubits are compressed onto higher dimensional qudits, so that extra space on the qudits can be used as ancillas for other purposes when computing is not in progress. This can be reversibly, so the qudits can be transformed back into qubits. In reference [16], it is mentioned that an *x-y-z qubit-qudit compression scheme* has been proposed.

However, the detailed scheme is complicated, so we will illustrate the main idea with a simple example of *qubit-qutrit compression*. Consider three qubits with $d = 2$ and two qutrits with $d = 3$. Three qubits can store a total of 2^3 computational states, thus have a state space of size 8, while two qudits have a state space with size $3^2 = 9$. Then it is possible to compress three qubits into two qudits and retrieve them using a compression/decompression circuit (figure 9).

Implementation. Practical realization is difficult due to limited support in mainstream quantum programming frameworks. As in the case of qutrits mentioned in the bucket-brigade scheme (subsection 3.1), physical implementation of qudits finds its way on physical quantum systems that have an *infinite spectrum of states*, like superconducting qubits and trapped ion qubits. In such systems, we identify different energy levels with different states of the quantum system.

2.6 Parameterized Quantum Circuits / EQGAN QRAM

Instead of circuit-based routing, PQC-based QRAM and EQGAN (Enhanced Quantum Generative Adversarial Network) models use trained quantum circuits to approximate memory access behavior. These are highly efficient in both depth and width—$O(1)$ and $O(n)$, respectively—making them ideal for near-term quantum devices.

In the section that follows, we deliver a simple error analysis for various archiectures, and showcase a Qiskit realization of Grover's search algorithm on the bucket-brigade and the flip-flop QRAM schemes.

3 Error Analysis

3.1 Bucket–Brigade QRAM

Let each routing node be a qutrit $\{|0\rangle, |1\rangle, |\bullet\rangle\}$. During one query exactly n nodes are driven by the address photons. Following the approach in Arunachalam *et al.* [**Arunachalam2015**], we assume each active routing node is affected by a local, independent CPTP noise channel, which we approximate as a depolarising channel for analytical tractability. Three logical outcomes may occur:

1. **Right path**. All n nodes switch correctly. Probability
$$P_{\text{right}} = (1-p)^n.$$
2. **Wrong path.** Exactly one active node flips to the wrong branch while the others work. The bus qubit still reaches a leaf but addresses a wrong memory cell, corrupting the data oracle. To first order in p
$$P_{\text{wrong}} \approx n\,p\,(1-p)^{n-1} \leq np.$$
3. **No path.** Any combination of two or more failed routers disconnects the tree, leaving the bus qubit in a dangling wave-guide. This event dominates the residual probability: $P_{\text{nop}} \approx 1 - P_{\text{right}} - P_{\text{wrong}}$.

A single call therefore has fidelity $F = P_{\text{right}} \simeq 1 - np$. Grover search makes $K = \Theta(2^{n/2})$ calls, so the overall success probability is $F^K \simeq \exp\bigl(-np\,2^{n/2}\bigr)$. Keeping that constant demands

$$p = \mathcal{O}\bigl(2^{-n/2}\bigr). \tag{3}$$

Correlated dephasing or photon-loss leakage changes only the constant prefactor [14]. Crucially, (3) shrinks *exponentially* with the address size, so bb-QRAM is unlikely to scale beyond $n \approx 10$ without full quantum error correction.

3.2 Fan–Out QRAM

A fan-out tree entangles the address register with a single bus photon that propagates through an $(2^n - 1)$-element GHZ state of routers [**Giovannetti2008**]. Modelling every router with an open-system master equation is intractable because the Lindblad operators act *non-locally* on the collective GHZ mode. The accepted workaround is a union-bound argument:

- If an independent Pauli error hits *any* router with probability p, coherence between two branches of the bus photon is destroyed.
- The probability that *no* router fails is $P_{\text{right}} = (1-p)^{2^n-1}$.

Demanding $P_{\text{right}} \geq 1 - \varepsilon$ yields

$$p \leq \frac{\varepsilon}{2^n - 1} = \mathcal{O}(2^{-n}). \tag{4}$$

Because the failure of *one* router already reveals which-path information, (4) is strictly tighter than the bucket-brigade bound (3). Inside Grover the requirement tightens to $p = \mathcal{O}(2^{-3n/2})$. Hence fan-out QRAM is even more noise-intolerant than bb-QRAM, a conclusion echoed by recent architectural surveys [**Phalak2023**, 13].

3.3 Flip–Flop QRAM

Flip-flop (FF) QRAM is implemented by a *circuit* that writes M classical n-bit words into amplitude encoding using $\Theta(M\,2^n)$ elementary gates [**Park2019**]. Park, Petruccione and Rhee assume that after every logical time step each qubit is depolarised with probability ε. Writing one data set therefore succeeds with probability

$$P_{\text{succ}} = (1-\varepsilon)^D, \quad D = \Theta(M\,n\log n), \tag{5}$$

where the $\log n$ term is the T-count overhead of decomposing an n-control rotation. Solving (5) for ε and expanding gives

$$\varepsilon = \mathcal{O}\bigl(1/(Mn\log n)\bigr).$$

Thus FF-QRAM tolerates *inverse-polynomial* gate noise when the state is prepared only once. If that same circuit is queried $K = \Theta(2^{n/2})$ times, one multiplies the error budget by K, reproducing an exponential constraint akin to (3). Because the register width is merely $n + m + 1$ qubits, however, the entire FF-QRAM can be encoded in a surface code, making the design far more fault-tolerance-friendly than bb- or fan-out QRAM.

3.4 Parameterized-Circuit QRAM (PQC-QRAM)

A newer line of work replaces large, deterministic loaders by shallow *variational* state-preparation circuits [1, 2]. Given classical amplitudes $\mathbf{v} \in \mathbb{R}^N$ one optimises a logarithmic-depth unitary $U(\boldsymbol{\theta})$ such that $U(\boldsymbol{\theta})\,|0\rangle^{\otimes n} \approx |v\rangle$.

Error sources. Two contributions add in series:

1. **Training error** $\varepsilon_{\text{train}} = 1 - |\langle v|U(\theta)0^{\otimes n}|v|U(\theta)0^{\otimes n}\rangle|^2$. This is a *classical* approximation error that can be made arbitrarily small given enough optimiser iterations [2].
2. **Hardware noise** Each of the $D = \Theta(\mathsf{poly}(n))$ gates undergoes a Pauli channel with probability p_g. The state fidelity becomes $F \approx (1 - p_g)^D$.

For an algorithm that calls the loader K times we demand $F^K(1 - \varepsilon_{\text{train}}) \geq 1 - \varepsilon$, giving

$$p_g \leq \frac{\varepsilon - \varepsilon_{\text{train}}}{K\,D} = \mathcal{O}\Big(\tfrac{1}{\mathsf{poly}(n)}\Big), \tag{6}$$

because both K and D are polynomial in n for all known PQC-QRAM proposals. Equation (6) is *polynomially* less stringent than the exponential bounds for router-based QRAMs, explaining why PQC-based loaders are currently regarded as the most NISQ-friendly memory interface [4]. They sacrifice the strict $O(1)$ query depth of true QRAM for trainability and noise resilience, a trade-off acceptable in many variational or sampling-based workloads.

4 Application of QRAM

4.1 Grover's Algorithm and QRAM

Quantum Random Access Memory (QRAM) enables a database to be queried in superposition by routing all address qubits at once through a shallow, reversible network. This reduces each oracle call in Grover's search to logarithmic depth, rather than linear depth of naive look-up, so that overall runtime remains dominated by algorithm $\mathcal{O}(\sqrt{N})$ iterations. In practice, one forwards the address register through QRAM to load the data, applies a phase flip on the marked element, and then uncomputes the route to erase entanglement, preserving coherence throughout amplitude amplification.

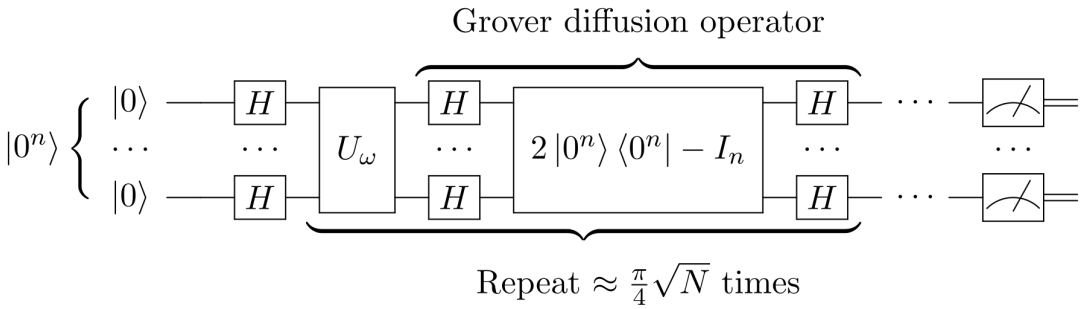

Figure 10: Schematic of Grover's algorithm with an oracle.

To implement the oracle in Grover's algorithm using QRAM, we perform the following steps:

1. **Load QRAM:** Route the address register through the QRAM network to retrieve the data value into an auxiliary / data register.
2. **Apply Phase Flip:** Conditioned on the data register that matches the target value, flip the phase of the joint state.
3. **Unload QRAM:** Reverse the routing path to uncompute the QRAM switches, disentangling the address bus while preserving the phase marking on the target.

Next, we will use the theoretical framework described above to implement both bucket-brigade and flip-flop QRAM architectures within the context of Grover's algorithm using the Qiskit library.

4.2 Bucket-Brigade QRAM in Grover Algorithm

Environment Setup and Package Imports.

```
from qiskit.circuit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.visualization import plot_histogram
from qiskit.compiler import transpile
from qiskit_aer import AerSimulator
import numpy as np, random, math
import pkg_resources
```

The following libraries are used to construct, execute, and analyze a Grover's algorithm simulation in Qiskit:

- `QuantumCircuit, QuantumRegister, ClassicalRegister` —Core Qiskit classes for building the quantum circuit: `QuantumRegister` holds the address and data qubits, `ClassicalRegister` stores measurement outcomes, and `QuantumCircuit` ties them together into a runnable circuit.
- `plot_histogram` —Visualization utility to display measurement statistics (probability distribution over outputs) and verify that the marked state is amplified.
- `transpile` —Optimizes and maps the abstract circuit onto the target simulator's basis gates, improving depth and performance.
- `AerSimulator` —QiskitAer backend for high-performance statevector or shot-based simulation, allowing us to run the Grover circuit with or without noise.

Parameters and Random Target.

```
n = 3
N = 2**n
target_index = random.randrange(N)
memory = [0]*N
memory[target_index] = 1
target_str = format(target_index, f'0{n}b')
```

This code randomly selects a target index from the 2^n possible states of an n-qubit system and marks it in the classical memory array.

Quantum Register Declaration.

```
addr   = QuantumRegister(n, 'addr')
router = QuantumRegister(N, 'router')
anc    = QuantumRegister(1, 'anc')
data   = QuantumRegister(1, 'data')
cl     = ClassicalRegister(n, 'cl')
qc = QuantumCircuit(addr, router, anc, data, cl)
```

This step allocates and bundles all registers required for the QRAM oracle:

- `addr`: n qubits holding the address to query.
- `router`: 2^n qubits forming the decoding network.
- `anc`: one ancilla qubit for implementing controlled operations.
- `data`: one qubit to load the memory value for phase marking.
- `cl`: n classical bits to record the final measured address.

QRAM Oracle Architecture.

```
# Step 1: Apply Hadamard on address register to create superposition
qc.h(addr)

# Step 2: Build bucket—brigade decoder network
for i in range(N):
    bin_address = format(i, f'0{n}b')
    # Encode address bits into router control lines
    for bit_idx, bit_val in enumerate(bin_address):
        if bit_val == '0':
            qc.x(addr[bit_idx])
    qc.mcx(addr, router[i])       # Multi-controlled X to set router[i]
    # Restore original address bits
    for bit_idx, bit_val in enumerate(bin_address):
        if bit_val == '0':
            qc.x(addr[bit_idx])

# Step 3: Load data into the data qubit using router lines
for i in range(N):
    if memory[i] == 1:
        qc.cx(router[i], data[0])

# Oracle: apply phase flip on the data qubit
qc.z(data[0])

# Step 4: Unload data (reverse of Step 3)
for i in reversed(range(N)):
    if memory[i] == 1:
        qc.cx(router[i], data[0])

```

```
# Step 5: Reset router network (reverse of Step 2)
for i in reversed(range(N)):
    bin_address = format(i, f'0{n}b')
    for bit_idx, bit_val in enumerate(bin_address):
        if bit_val == '0':
            qc.x(addr[bit_idx])
    qc.mcx(addr, router[i])
    for bit_idx, bit_val in enumerate(bin_address):
        if bit_val == '0':
            qc.x(addr[bit_idx])
```

- **Hadamard on `addr`** prepares all 2^n addresses in superposition.
- **Decoder network (router)**: for each address i, conditional MCX gates route the address path into the corresponding `router[i]` qubit.
- **Data load**: controlled-X gates from active router lines fetch the stored bit into `data`.
- **Oracle phase flip**: a Z gate flips the phase of the marked state on `data`.
- **Data unload**: reverse the loading step to uncompute data mapping.
- **Router reset**: reverse the decoder construction to clear entanglement in the router network.

Diffusion Operator.

```
# Diffusion (inversion-about-the-mean) operator on addr register
qc.h(addr)
qc.x(addr)
qc.h(addr[-1])
qc.mcx(addr[:-1], addr[-1], ancilla_qubits=[anc[0]])
qc.h(addr[-1])
qc.x(addr)
qc.h(addr)
```

The diffusion operator serves to amplify the amplitude of the marked state after the oracle has applied a phase flip.

Measurement. After the oracle and diffusion steps, we perform a measurement of the address register to read out the result:

```
# Collapse the quantum state into classical bits
qc.measure(addr, cl)
```

This instruction maps the superposed amplitudes of `addr` into classical outcomes `cl`, yielding one of the 2^n basis states with probabilities determined by the amplitude distribution.

To verify our analytic prediction and collect statistics, we run the circuit multiple times:

```
# Number of shots for empirical measurement
small_shots = 10000
sim_small   = AerSimulator()

# Copy the circuit and ensure all qubits are measured
qc_small    = qc.copy()
qc_small.measure_all()

# Execute and retrieve counts
result = sim_small.run(qc_small, shots=small_shots).result()
counts = result.get_counts()
```

Finally, we visualize the frequency of each outcome using a histogram.

```
plot_histogram(counts)
```

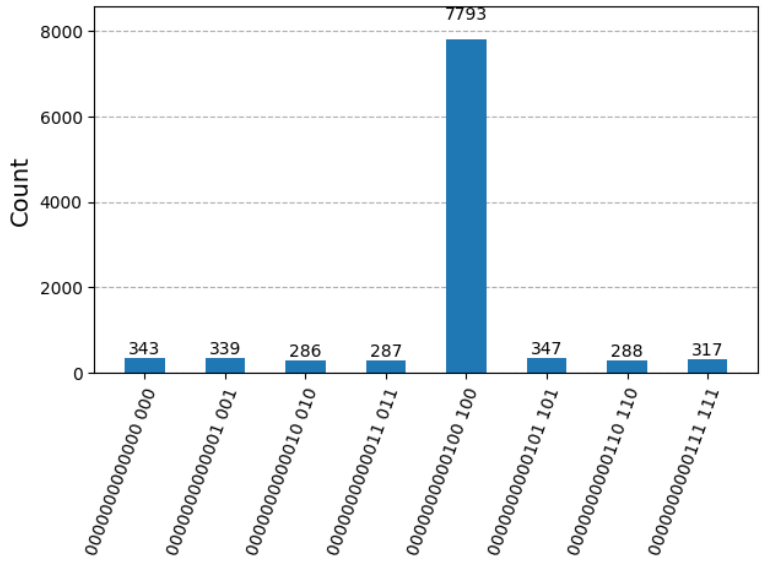

Figure 11: Measurement of bucket brigade n=3 when only one iteration

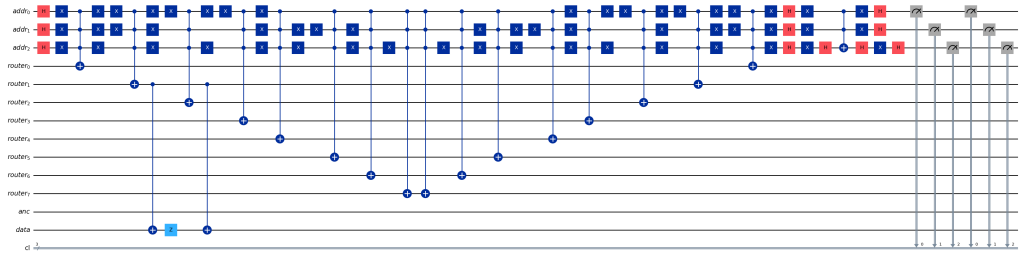

Figure 12: bucket brigade n=3 circuit

The resulting histogram shows a clear peak in the marked state, confirming that the Grover oracle plus diffusion successfully amplifies the probability of the target above the uniform background.

For $n = 3$ qubits, the search space size is

$$N = 2^3 = 8.$$

The fundamental angle is

$$\theta_0 = \arcsin\big(1/\sqrt{N}\big) = \arcsin\big(1/\sqrt{8}\big) \approx 0.3614.$$

The optimal number of Grover iterations is

$$k_{\text{opt}} = \left\lfloor \frac{\pi}{4\,\theta_0} \right\rfloor = \left\lfloor \frac{\pi}{4 \times 0.3614} \right\rfloor = 2.$$

The maximum success probability is

$$P_{\text{theory}} = \sin^2\big((2\,k_{\text{opt}} + 1)\,\theta_0\big) = \sin^2(5 \times 0.3614) \approx 0.9453 \quad (\approx 94.5\%).$$

Therefore, after the initial Hadamard layer in the address register, one should repeat the following two-step Grover iteration exactly $k_{\text{opt}} = 2$ times:

1. **Quantum Oracle:** (QRAM load phase flip QRAM unload)
2. **Diffusion Operator:** inversion about the mean on the address register

This achieves the highest probability of measuring the marked state for $n = 3$.

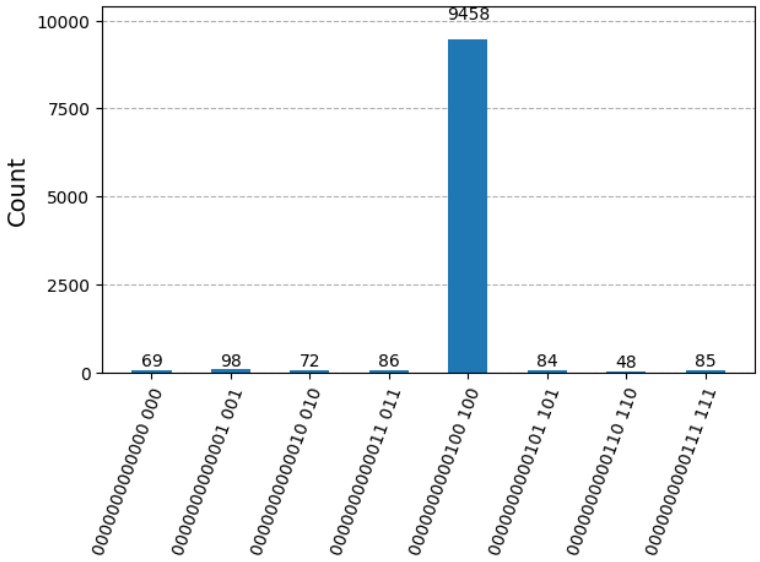

Figure 13: bucket brigade n=3 achieves the highest probability after two iterations

4.3 Flip-Flop QRAM in Grover Algorithm

Quantum Register Declaration.

```
addr = QuantumRegister(n, 'addr')    # address register
data = QuantumRegister(1, 'data')    # data qubit for fetched value
anc  = QuantumRegister(1, 'anc')     # ancilla for conditional phase flip
cl   = ClassicalRegister(n, 'cl')    # classical bits for measurement
qc   = QuantumCircuit(addr, data, anc, cl)
```

Unlike the QRAM bucket bridge, which allocates a full `router` register of size 2^n for its switching network, the flip—flop design embeds the routing logic directly into controlled gates in the `addr` and `data` qubits. This eliminates the need for a large router register, reducing qubit overhead at the expense of a deeper, sequential gate structure.

QRAM Oracle Architecture. This implementation uses a flip-flop-style QRAM oracle, without a large router register, by matching each address pattern directly on the data qubit:

```
# Step 1: Create superposition on address register
qc.h(addr)

# Step 2: Encode address and load memory into data qubit
thetas = [math.pi if memory[j]==1 else 0 for j in range(N)]
for j, theta in enumerate(thetas):
    for bit in range(n):
        if ((j >> bit) & 1) == 0:
            qc.x(addr[bit])
    qc.mcx(addr, anc[0], ancilla_qubits=[])
    qc.cry(theta, anc[0], data[0])
    qc.mcx(addr, anc[0], ancilla_qubits=[])
    for bit in range(n):
        if ((j >> bit) & 1) == 0:
            qc.x(addr[bit])

# Oracle: phase flip on the loaded data qubit
qc.z(data[0])

# Step 3: Unload data by reversing Step 2
for j, theta in reversed(list(enumerate(thetas))):
    for bit in range(n):
```

```
        if ((j >> bit) & 1) == 0:
            qc.x(addr[bit])
    qc.mcx(addr, anc[0], ancilla_qubits=[])
    qc.cry(-theta, anc[0], data[0])
    qc.mcx(addr, anc[0], ancilla_qubits=[])
    for bit in range(n):
        if ((j >> bit) & 1) == 0:
            qc.x(addr[bit])
```

- **No dedicated router register:** Unlike the bucket-brigade tree (which uses a 2^n-qubit `router` register to fan out address lines), flip—flop applies multi-controlled gates directly on the `addr` and `data` qubits.
- **Sequential control flow:** Each address bit pattern is processed in sequence: encode, fetch, phase flip, and uncompute, resulting in a deeper circuit but lower qubit count.
- **Ancilla reuse:** A single ancilla qubit (`anc`) is reused for all multi-controlled operations, instead of per-node storage.

This design trades off increased circuit depth and gate count for reduced qubit overhead, making it suitable when qubit resources are more constrained than coherence time.

Measurement. After running the oracle and diffuser, we measure the `addr` register on the computational basis. This collapses the superposition into a single basis state, with probabilities given by the amplified amplitudes.

Because both the bucket—brigade and flip—flop QRAM implementations perform the same Grover iterations (oracle load phase flip unload diffusion) in the address register, they yield identical measurement statistics. In particular, both architectures achieve the same optimal success probability.

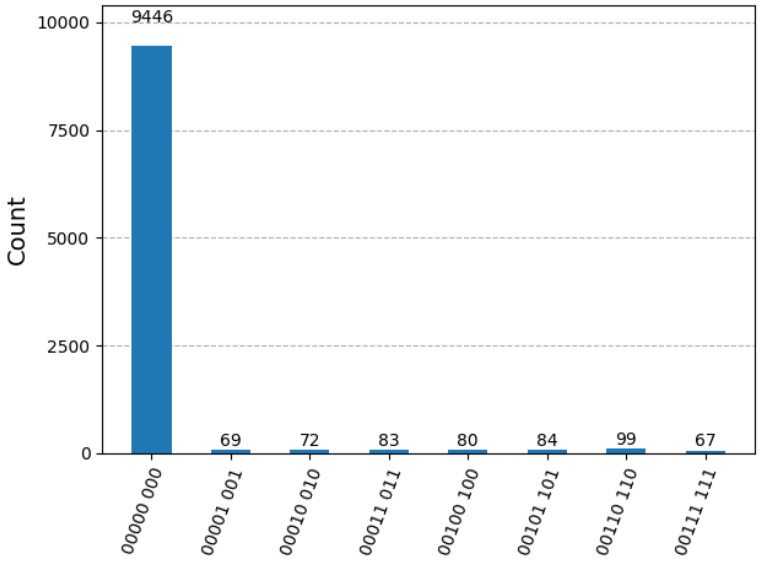

Figure 14: flip-flop n=3 achieves the highest probability after two iterations

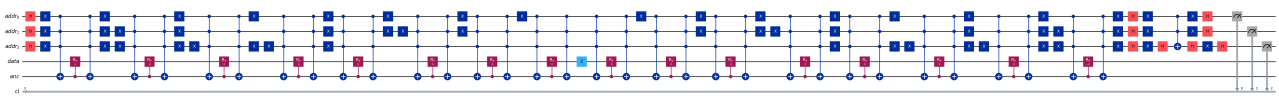

Figure 15: flip-flop n=3 circuit

5 Conclusion and Future Work

5.1 Summary of Results

We have discussed various architectures for QRAM.

5.2 Future Study Directions

We may expand on our QRAM simulations by incorporating, for example, tensor networks. network）

6 Task Distribution

Section	Contributor
Introduction	Yu-Cheng Liu
Architecture	Shao-Kai Huang
Error Analysis	Yu-Cheng Liu
Applications (and Quiskit code)	Yu-Lun Chen
Conclusion	Shao-Kai Huang

Table 2: Contributors for each section in the written report. The corresponding sections in the oral presentation are also prepared and delivered by the same authors.

References

[1] Marcello Benedetti et al. “Parameterized Quantum Circuits as Universal Function Approximators”. In: *Quantum Sci. Technol.* 4 (2019), p. 043001.

[2] Yuxuan Du, Min-Hsiu Hsieh, and Tongliang Liu. “Efficient Quantum Amplitude Encoding of Data”. In: *npj Quantum Information* 8 (2022), p. 85.

[3] Christoph Dürr and Peter Høyer. *A Quantum Algorithm for Finding the Minimum*. arXiv:quant-ph/9607014. 1996.

[4] András Gilyén, Rui B. Gomes, and Arthur Jacquard. *Data-Equivariant Variational Quantum Algorithms with QRAM Initialisation*. arXiv:2301.07775. 2023.

[5] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. “Quantum Random Access Memory”. In: *Phys. Rev. Lett.* 100 (16 Apr. 2008), p. 160501. DOI: 10.1103/PhysRevLett.100.160501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.100.160501.

[6] Lov K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *Proceedings of the 28th Annual ACM Symposium on Theory of Computing* (1996), pp. 212–219.

[7] A. W. Harrow, A. Hassidim, and S. Lloyd. “Quantum Algorithm for Linear Systems of Equations”. In: *Phys. Rev. Lett.* 103 (2009), p. 150502.

[8] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. 6th ed. Morgan Kaufmann, 2017.

[9] Bruce Jacob, Spencer Ng, and David Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2007.

[10] Ken M. Kaneko, Guillaume Gautier, and Stefan Woerner. “Quantum Amplitude Estimation under Realistic Noise”. In: *Advances in Financial Machine Learning* (2020). arXiv:2009.04471.

[11] Iordanis Kerenidis and Anupam Prakash. “Quantum Recommendation Systems”. In: *Proceedings of ITCS 2017*. arXiv:1603.08675. 2017.

[12] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. “Quantum Principal Component Analysis”. In: *Nature Physics* 10 (2014), pp. 631–633.

[13] Luis Morales and Nicolas Delfosse. *Architectural Considerations for Quantum Random Access Memories*. arXiv:2310.04567. 2023.

[14] Eliana Moreira and Anupam Prakash. *Correlated Noise in Bucket-Brigade QRAM*. arXiv:2403.01234. 2024.

[15] Kazuki Nakaji and Kosuke Mitarai. "Faster Quantum Global Minimum Search by Incremental QRAM Updating". In: *QIP 2021*. arXiv:2009.04451. 2021.

[16] Koustubh Phalak, Avimita Chatterjee, and Swaroop Ghosh. "Quantum Random Access Memory for Dummies". In: *Sensors* 23.17 (2023). ISSN: 1424-8220. DOI: 10.3390/s23177462. URL: https://www.mdpi.com/1424-8220/23/17/7462.

[17] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. "Quantum Support Vector Machine for Big Data Classification". In: *Phys. Rev. Lett.* 113 (2014), p. 130503.

[18] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. Springer, 2019.

[19] Yuchen Wang et al. "Qudits and High-Dimensional Quantum Computing". In: *Frontiers in Physics* Volume 8 - 2020 (2020). ISSN: 2296-424X. DOI: 10.3389/fphy.2020.589504. URL: https://www.frontiersin.org/journals/physics/articles/10.3389/fphy.2020.589504.

[20] Nathan Wiebe, Ashish Kapoor, and Krysta Svore. "Quantum Algorithms for Nearest-Neighbour Methods for Supervised and Unsupervised Learning". In: *Quantum Information & Computation* 15.3–4 (2015), pp. 318–358.

[21] Lei Zhang, Haonan Sun, et al. *Cryogenic CMOS Interfaces for Scalable Quantum Memories*. arXiv:2402.12345. 2024.

[22] Christa Zoufal and Stefan Woerner. "Variance-Reduced Quantum k-Nearest Neighbour Classification". In: *npj Quantum Information* 8 (2022), p. 93.