Master QLMN
Experimental quantum computing

François-Marie Le Régent
francois-marie.le-regent@pasqal.com

Lecture 6 - Grover's Algorithm, Quantum Fourier Transform and Resource Estimates

February 4th 2026

1 Introduction

In the first part of this course, we explored the physical implementations of quantum computing: neutral atoms in optical tweezers, trapped ions, superconducting circuits cooled to millikelvin temperatures, photons, and electron spins in quantum dots. Each platform offers different engineering trade-offs, coherence times, and scalability prospects.

Last week, we took our first deep dive into quantum algorithms with **Shor's factoring algorithm**, the poster child for exponential quantum speedup. We saw how factoring an integer N reduces to the problem of finding the period r of the function $f(x) = a^x \bmod N$, and how quantum computers can solve this period-finding problem *exponentially* faster than any known classical algorithm. Along the way, we briefly mentioned two powerful quantum subroutines that made this possible: the **Quantum Fourier Transform (QFT)** and **Quantum Phase Estimation (QPE)**. In that lecture, we treated QFT and QPE primarily as black box tools that we plugged into Shor's algorithm to extract the period. We didn't fully unpack *how* they worked or *why* they're so powerful beyond this specific application. Today's lecture aims at opening those black boxes, as well as another well know quantum algorithm, and understand the quantum primitives that power not just Shor's algorithm, but the landscape of quantum algorithms:

- **Grover's Algorithm:** Last week we mentioned that Grover provides (only) a *quadratic* speedup (vs Shor's exponential) for unstructured search. We'll see exactly how **amplitude amplification** works by relying on a geometric rotation in Hilbert space.
- **Quantum Fourier Transform (QFT):** We'll go beyond using QFT as a subroutine and understand its circuit implementation, its connection to classical Fourier analysis, and why it achieves an exponential speedup.
- **Quantum Phase Estimation (QPE):** We'll formalize this eigenvalue extraction technique and see why it's the workhorse behind not just Shor's algorithm, but also quantum chemistry simulations, solving linear systems, and many other applications.

Beyond understanding these individual algorithms in depth, we'll step back to survey the broader (but partial) **quantum algorithm landscape**: What classes of problems admit quantum speedups? Where are the boundaries? Which real-world applications might actually benefit from quantum hardware in the near term?

Finally, we'll confront the **resource estimation reality check**. Shor's algorithm can in theory factor a 2048-bit RSA key but how many physical qubits does that actually require? How long would it take? What error rates can we tolerate? These aren't just academic questions but determine whether quantum advantage is achievable with current or near-term hardware. **Spoiler alert:** the resource estimates will reveal that we cannot run most of these algorithms on today's noisy devices without bridging the gap between current machines' error rates vs what is required for high-level applications. This sets the floor for the **quantum error correction** lectures that will follow, where we'll see exactly how to bridge this gap between the imperfect physical qubits from our first lectures and the pristine logical qubits that algorithms require. Error correction isn't a just nice-to-have feature, it's the essential bridge from quantum computing theory to quantum computing reality.

For next time 2026-02-11

You are not expected to solve any exercice from today's lecture. Instead, read Chen et al, *Exponential suppression of bit or phase errors with cyclic error correction* , Nature 2021 (https://www.nature.com/articles/s41586-021-03588-y) [1]. We'll discuss it during next lecture.

Let's begin by diving into Grover's algorithm, the quintessential example of quadratic quantum speedup.

2 Grover's Algorithm

2.1 Quizz

Last time, we asked you to read the paper [2]. Let's test our understanding of this pioneering (from 1998!) NMR implementation of Grover's algorithm with some questions covering the physical implementation, experimental techniques, and the specific dynamics of the $N = 4$ case.

Q1: Physical Hamiltonian and Qubit Encoding

Regarding the physical Hamiltonian and qubit encoding used in this experiment *(select all correct answers)*:
A. The experiment was conducted using a homonuclear spin system at cryogenic temperatures.
B. The two qubits were represented by the nuclear spins of Hydrogen (^{1}H) and Carbon-13 (^{13}C) in chloroform molecules.
C. The interaction term in the Hamiltonian responsible for creating entanglement is of the Ising form $H_{int} \propto I_{zA} I_{zB}$.
D. The scalar coupling constant J was approximately 215 Hz, allowing for gate operations in the millisecond regime.

Q2: Pure State Preparation

How was the "pure" initial state $|00\rangle$ prepared from the thermal equilibrium state *(select all correct answers)*?
A. The sample was cooled to the ground state using optical pumping.
B. A technique called "temporal labeling" was used, which involves summing the results of three separate experiments with permuted populations.
C. The experiment utilized a single-shot projective measurement to collapse the ensemble into the ground state.
D. The effective pure state was defined using deviation density matrices, where the identity part of the thermal state is ignored as it does not contribute to the NMR signal.

Q3: Grover Operator and Oracle Implementation

Regarding the implementation of the Grover operator U and the Oracle *(select all correct answers)*:
A. The conditional phase shift (the Oracle) was implemented using the natural scalar coupling evolution of the spins ($2\pi J I_{zA} I_{zB} t$).
B. For the $N = 4$ case, the algorithm theoretically requires approximately $\frac{\pi}{4}\sqrt{N} \approx 1.57$ iterations to maximize probability.
C. The "inversion about the mean" operator D was compiled into a pulse sequence represented as $D = WPW$.
D. The Walsh-Hadamard transform was applied to both spins using the optimized pulse sequence $H = X\bar{Y}_{\frac{1}{2}}$ (rotation about X after a rotation about -Y).

Q4: Algorithm Dynamics and Periodicity

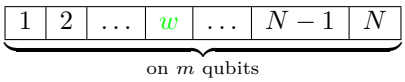
What do the experimental results demonstrate regarding the algorithm's dynamics and periodicity *(select all correct answers)*?
A. The amplitude of the solution state $|11\rangle$ (for marked state $x_0 = 3$) reaches a maximum after exactly 1 iteration.
B. The density matrix exhibits a periodicity of 3 iterations, consistent with the theoretical prediction for $N = 4$.
C. The experiment showed that the algorithm is unstable and loses periodicity after the first cycle due to decoherence.
D. The classical expected number of queries for this search is 2.25, whereas the quantum implementation solved it in a single step.

Q5: Read-out Method and Error Analysis

Regarding the read-out method and error analysis *(select all correct answers)*:
A. The final state was determined by full state tomography, which required reconstructing the density matrix from 9 different rotation experiments.
B. The longest computation (7 iterations) took approximately 35ms, which was significantly longer than the coherence time T_2.
C. The signal strength in the NMR spectrum represents the fraction of the population in a given state, rather than a collapse to an eigenstate.
D. The primary source of experimental error (7-44%) was identified as the inhomogeneity of the magnetic field.

2.2 Introduction

Grover's algorithm addresses one of the most fundamental problems in computer science: searching through an unstructured database. Let's start by explaining the problem at stake. Imagine you have a list of $N = 2^m$ items, and somewhere in that list is a special element that we call w (aka the "winner") that satisfies some unique property. Classically, finding this needle in items haystack requires checking them one by one: on average $N/2$ queries, and in the worst case, all N items.



Grover's algorithm [3] provides a quadratic speedup, finding the winner in only $O(\sqrt{N})$ queries. While this may seem modest compared to the exponential speedup of Shor's algorithm, Grover's approach is widely applicable which make it so valuable. The underlying technique of **amplitude amplification** serves as a general subroutine for any problem where solutions can be verified efficiently.

2.3 Algorithm Overview

The algorithm consists of three main components that work together to concentrate probability amplitude on the winning state:

- **State Preparation:** We initialize the quantum register in a uniform superposition over all possible items in our search space.
- **Oracle:** A black-box operation that "marks" the correct answer by flipping its phase (without revealing which state it is).
- **Diffusion Operator:** An amplification step that increases the probability of measuring the marked state while suppressing all others.

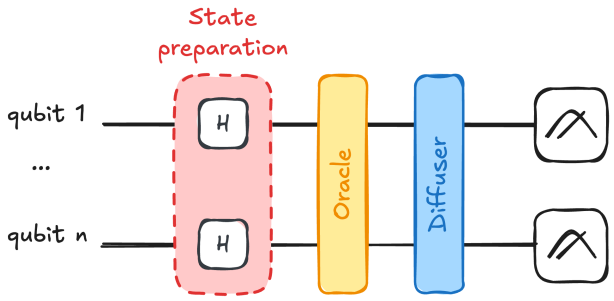


Figure 1: Grover's algorithm circuit showing initialization, oracle, and diffusion operator

See fig. 1 for the circuit structure for Grover's algorithm. The circuit begins by the state preparation phase in red with Hadamard gates creating the uniform superposition, followed by the oracle (yellow) marking the winner state, and then the diffusion operator (blue) that amplifies the marked state's amplitude. For small databases like $N = 4$ as we will see later, a single iteration achieves near-perfect success probability. For other settings, we need to repeat Oracle+Diffuser. Let's go through all the steps of this algorithm one by one.

2.3.1 The Initial Superposition

Since we have no prior knowledge about which item is the winner, we begin by preparing a uniform superposition over all N basis states. This is achieved by applying Hadamard gates, introduced in the first lecture, to each of the m qubits initialized to $|0\rangle$:

$$|s\rangle = H^{\otimes m}|0\rangle^{\otimes m} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

At this stage, if we were to measure immediately, each outcome $|x\rangle$ would appear with equal probability $1/N$. The probability of successfully finding the winner in a single shot is therefore $1/N$, no better than classical random guessing. To see why this is problematic, consider that the expected number of queries X to find the item classically is:

$$\mathbb{E}(X) = \sum_{k=1}^{N} k \cdot \mathbb{P}(X = k) = \sum_{k=1}^{N} k \cdot \frac{1}{N} = \frac{1}{N} \frac{N(N+1)}{2} = \frac{N+1}{2}$$

Our goal is thus to transform this uniform distribution into one heavily concentrated on $|w\rangle$, so that measurement yields the correct answer with high probability after only $O(\sqrt{N})$ iterations.

2.3.2 The Oracle or the Yes/No Verifier Blackbox Function

Grover's Algorithm is a verification algorithm

We emphasize that Grover's algorithm does not search through items one by one. Instead, it assumes we already have access to a function that can *verify* whether a given input is the solution, and uses quantum interference to amplify the correct answer.

Grover's algorithm does not "look inside" the database. Rather, it relies on an **oracle**, a black-box function that can recognize the winner when presented with it. The oracle's job is simple: given any state $|x\rangle$, flip its phase if and only if $x = w$.

This distinction between *finding* and *verifying* is fundamental in complexity theory. Recall from lecture 1 that many problems in the class **NP** (Non-deterministic Polynomial time) are hard to solve but easy to verify.

One example to understand this subtle difference is to consider Sudoku. Checking whether a completed grid is valid takes seconds, but finding the solution from scratch can be much harder. Grover's algorithm exploits this asymmetry, working for any problem where verification is efficient. To get some intuition of what the oracle is, let's go through multiple definitions and interpretations.

2.3.2.1 Mathematical Definition of the Oracle As we just mentioned, the oracle U_w acts on computational basis states as follows:

$$U_w|x\rangle = \begin{cases} |x\rangle & \text{if } x \neq w \\ -|x\rangle & \text{if } x = w \end{cases}$$

This can be written compactly as $U_w = \mathbb{I} - 2|w\rangle\langle w|$, which is a reflection operator about the vector state $|w\rangle$. In matrix form, for a 2-qubit system where the winner is $|10\rangle$, we can write:

$$U_{|10\rangle} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Notice that only the diagonal entry corresponding to the winner state carries a -1. This phase flip is *invisible* if you measure immediately (the probabilities |amplitude|^2 are unchanged), but it sets the stage for interference in the next step.

Exercise 1: Oracle Matrix Construction

For a 3-qubit system ($N = 8$), construct the explicit 8×8 oracle matrix U_w when the winner state is $|w\rangle = |101\rangle$. Verify that U_w is unitary by checking $U_w U_w^\dagger = I$.

Exercise 2: Expected Number of Classical Queries

Show that for a database of size N, the expected number of classical queries to find a unique item is $(N + 1)/2$. What is the variance of the number of queries?

2.3.2.2 Building the Oracle from a Classical Function Another way to get intuition on this oracle function is to express it from a classical function. Indeed, in practice, we construct the oracle from a classical boolean function $f(x)$ that identifies the winner:

$$f(x) = \begin{cases} 0 & \text{if } x \neq w \\ 1 & \text{if } x = w \end{cases}$$

The oracle's action can then be written as $U_w|x\rangle = (-1)^{f(x)}|x\rangle$. But how do we actually implement this phase flip using quantum gates?

The standard approach uses **phase kickback**. Suppose we have a quantum circuit U_f that computes f in the usual way: $|x\rangle|0\rangle \xrightarrow{U_f} |x\rangle|f(x)\rangle$. The trick is to prepare the target qubit not in $|0\rangle$, but in a particular state, for instance $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, that turns out to be an eigenstate of the unitary transformation. When we apply U_f with this special target state, the following happens. If $f(x) = 0$, the target remains $|-\rangle$. But if $f(x) = 1$, the XOR operation flips $|-\rangle$ to $-|-\rangle$, introducing a global phase of -1. The result is:

$$|x\rangle|-\rangle \xrightarrow{U_f} (-1)^{f(x)}|x\rangle|-\rangle$$

The phase has been "kicked back" onto the input register, while the target qubit is left unchanged (up to a phase we can ignore). This is the standard technique for converting any classical verification function into a quantum phase oracle. Let's explain it in more details.

2.3.2.3 Phase Kickback: The Underlying Oracle Mechanism

To understand this more generally, consider a controlled-U gate where the target is prepared in an eigenstate $|\psi\rangle$ satisfying $U|\psi\rangle = e^{i\phi}|\psi\rangle$.

Key Concept

Phase kickback occurs when a controlled operation acts on an eigenstate of the target unitary. The eigenvalue phase gets transferred to the control qubit's relative phase.

If the control qubit starts in a superposition:

$$C\text{-}U\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |\psi\rangle\right) = \frac{|0\rangle|\psi\rangle + e^{i\phi}|1\rangle|\psi\rangle}{\sqrt{2}} = \left(\frac{|0\rangle + e^{i\phi}|1\rangle}{\sqrt{2}}\right) \otimes |\psi\rangle$$

The target qubit remains in $|\psi\rangle$, but the control qubit has acquired a new relative phase $e^{i\phi}$. This mechanism allows us to extract information about U (its eigenvalue) and encode it as a phase on another register (here the control register), a principle that underlies not just Grover's oracle, but also quantum phase estimation and many other algorithms, we will thus mention it again below.

Exercise 3: Phase Kickback with Pauli Gates

Consider the controlled-Z gate acting on $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |1\rangle$. Show that the phase gets kicked back to the control qubit. What is the resulting state?

Exercise 4: Oracle Action on Superposition

Consider a 2-qubit system with winner $|w\rangle = |11\rangle$. Calculate $U_w|s\rangle$ explicitly, where $|s\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$.

2.4 Geometric Interpretation of Grover's Algorithm

We have not yet covered all the 3 steps of Grover's algorithm, we are still missing the Diffuser one. But before diving into this last part, we take a moment to focus on a geometric interpretation that will help us introduce this last part. Grover's algorithm is indeed best understood through geometry. The key insight that we will discover is that **two reflections compose to give a rotation**, a fact from elementary geometry that becomes the engine of Grover's quantum speedup.

See fig. 2 for an step-by-step illustration of how amplitudes evolve during Grover's algorithm. Before the oracle, all states have equal positive amplitude. After the oracle, the winner state's amplitude becomes negative, creating an asymmetry. The diffusion operator then reflects all amplitudes about their mean, dramatically boosting the winner's amplitude while slightly decreasing all others.

2.4.1 Setting Up the 2D Subspace

Let's focus on the top row of fig. 2. Although our Hilbert space has dimension $N = 2^m$, the entire algorithm takes place in a two-dimensional subspace spanned by just two states:

- $|w\rangle$: the target state we wish to find
- $|s'\rangle$: the uniform superposition over all *non-winner* states, defined as $|s'\rangle = \frac{1}{\sqrt{N-1}} \sum_{x \neq w} |x\rangle$

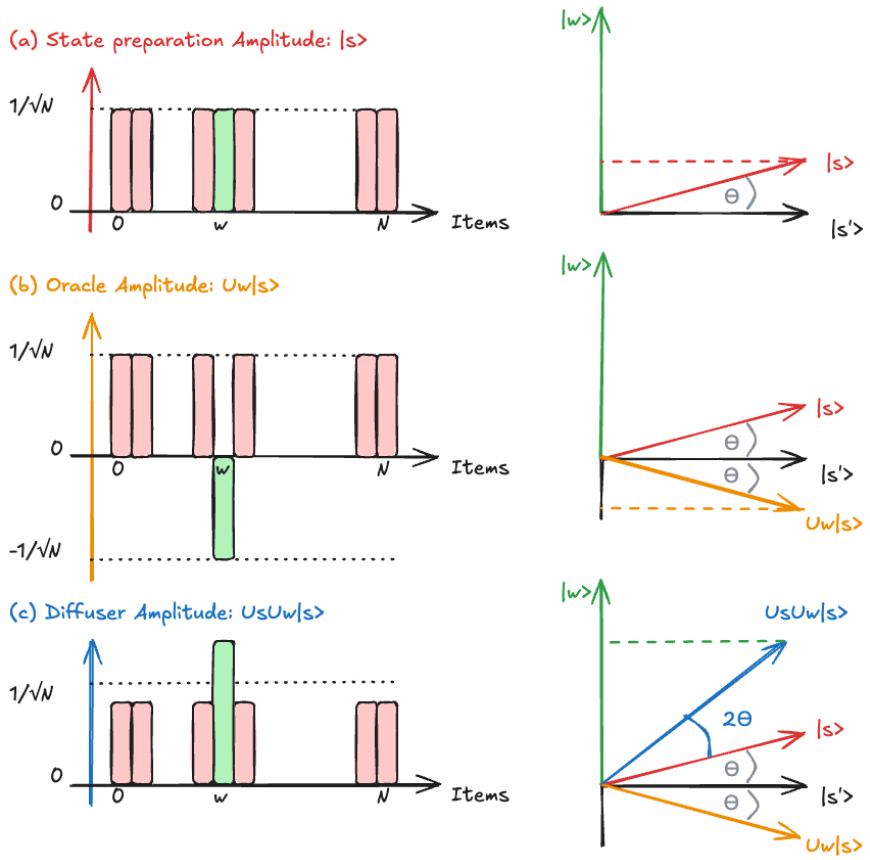


Figure 2: Amplitude evolution during Grover's algorithm showing state preparation (red), oracle (orange) and diffusion (blue) steps

The initial state $|s\rangle$ lies in this plane because it can be written as a linear combination of $|w\rangle$ and $|s'\rangle$. Since $|s\rangle$ has only a tiny component along $|w\rangle$ (amplitude $1/\sqrt{N}$), it starts almost parallel to $|s'\rangle$. More precisely, we can write any state in this subspace as:

$$|\psi\rangle = \sin\theta\,|w\rangle + \cos\theta\,|s'\rangle$$

For the initial state, the angle from the $|s'\rangle$ axis is:

$$\theta_0 = \arcsin\frac{1}{\sqrt{N}} \approx \frac{1}{\sqrt{N}} \quad \text{(for large } N\text{)}$$

This tiny angle (roughly $1/\sqrt{N}$ radians) is the starting point of our journey toward $|w\rangle$.

Exercise 5: Initial Angle Calculation

For $N = 16$ (4 qubits), calculate the exact initial angle $\theta_0 = \arcsin(1/\sqrt{N})$ in both radians and degrees. Compare this to the approximation $\theta_0 \approx 1/\sqrt{N}$.

2.4.2 Step 1: the Oracle as a Reflection

Now let's switch to the second row of fig. 2. We mentioned several interpretations for the oracle U_w, the final one we will use is its geometric one. The oracle $U_w = I - 2|w\rangle\langle w|$ performs a **reflection about the hyperplane perpendicular to** $|w\rangle$. Geometrically, if you picture $|w\rangle$ as the vertical axis in our 2D subspace, see fig. 2, the oracle flips the state vector across the horizontal axis $|s'\rangle$. The effect is simple: if the state was at angle θ above the $|s'\rangle$ axis, after the oracle it sits at angle $-\theta$ (below the axis). The distance to $|w\rangle$ hasn't changed, but the state has been "flipped" to the other side of $|s'\rangle$.

Exercise 6: Oracle Reflection Verification

Verify that $U_w|x\rangle = |x\rangle$ when $x \neq w$ and $U_w|w\rangle = -|w\rangle$ by direct calculation using $U_w = I - 2|w\rangle\langle w|$.

2.4.3 Step 2: The Diffusion Operator as a Second Reflection

Now we briefly mention the diffuser's geometric action to understand its effect intuitively. As can be seen in the last row of fig. 2, the diffuser is a second reflection, now performed about the initial state $|s\rangle$ itself.

Exercise 7: Diffusion Operator Matrix Form

For a 2-qubit system ($N = 4$), construct the explicit 4×4 diffusion operator matrix $U_s = 2|s\rangle\langle s| - I$, where $|s\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$.

2.5 The Diffusion Operator: Amplifying the Marked State

After applying the oracle, our quantum state looks similar to where we started, at least on the surface. Indeed, if we measured at this point, we would still get each outcome with equal probability, since $|1/\sqrt{N}|^2 = |-1/\sqrt{N}|^2$. But what is crucial is that every amplitude remains $1/\sqrt{N}$ in magnitude, except the winner, which now carries a negative sign!

State	$\|000\rangle$	$\|001\rangle$	...	$\|100\rangle$	$\|101\rangle$
Amp. Before Oracle	$1/\sqrt{N}$	$1/\sqrt{N}$	...	$1/\sqrt{N}$	$1/\sqrt{N}$
Amp. After Oracle	$1/\sqrt{N}$	$1/\sqrt{N}$	...	$-1/\sqrt{N}$	$1/\sqrt{N}$

State	$\|000\rangle$	$\|001\rangle$	...	$\|100\rangle$	$\|101\rangle$
Goal	0	0	...	1	0

The final step is the **diffusion operator** U_s, also known as the Grover diffusion or "inversion about the mean." As we just saw geometrically, this operator reflects all amplitudes about their average value $|s\rangle$, which has the effect of boosting the marked state (whose amplitude is below the mean) while suppressing all others. Mathematically, the diffusion operator is defined as:

$$U_s = 2|s\rangle\langle s| - I$$

where $|s\rangle$ is still the uniform superposition. The full Grover's algorithm is composed of the oracle and diffuser, but how many times? We repeat it to get as close as possible to the winner state $|w\rangle$.

Sign Convention

The difference in signs betweenn $U_w = I - 2|w\rangle\langle w|$ and $U_s = 2|s\rangle\langle s| - I$ comes from **which vector you use to define the reflection**.
Mathematically, both operators perform the exact same geometric action: **reflection about an axis**. The general formula for reflecting a vector across an axis defined by a projector P is:

$$\text{Reflection} = 2P - I$$

Here is why the signs look different for the two operators:
The Diffuser (U_s): Reflect the state about the axis **parallel** to $|s\rangle$. We have the projector for this axis: $P_s = |s\rangle\langle s|$. Using the standard formula:

$$U_s = 2P_s - I = 2|s\rangle\langle s| - I$$

(This is a standard reflection about the vector $|s\rangle$.)
The Oracle (U_ω): Reflect the state about the axis **orthogonal** to the winner $|\omega\rangle$ (i.e., the axis of "non-winners") that we called $|s'\rangle$. Its projector is $P_{s'}$. Using the standard formula, the operator should be:

$$U_\omega = 2P_{s'} - I$$

However, we don't build the circuit using "non-winners"; we build it using the winner $|\omega\rangle$. We know that the sum of projectors in the space must be the Identity:

$$P_{s'} + P_\omega = I \quad \Longrightarrow \quad P_{s'} = I - P_\omega$$

Substitute this into our reflection formula:

$$U_\omega = 2(I - P_\omega) - I = 2I - 2P_\omega - I = I - 2|\omega\rangle\langle\omega|$$

Summary: The sign flips because of the **basis** we use to write the operator. U_s is written using the axis we reflect **about** (positive P). U_ω is written using the axis **normal** to the reflection plane (negative P).

2.6 Iteration and Optimal Stopping

The following fig. 3 builds on fig. 2 and depicts the geometric view of the full Grover's algorithm in the 2D subspace spanned by $|w\rangle$ and $|s'\rangle$ where Oracle and Diffuser are repeated. Composing these two reflections produces a rotation by 2θ towards the target state $|w\rangle$. After k iterations, the state has rotated through angle

$(2k+1)\theta$. The amplitude of the winner state grows while all other amplitudes shrink, this is the amplification we sought.

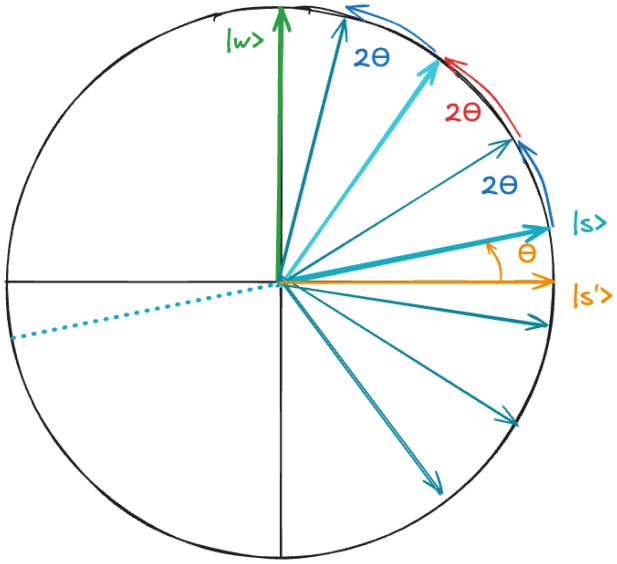


Figure 3: Geometric view of Grover's algorithm as rotations in 2D subspace

We want to stop when this angle reaches approximately $\pi/2$, at which point the state is nearly aligned with $|w\rangle$ and measurement succeeds with high probability. This can be see as $k^* = \lfloor \frac{\text{Total distance}}{\text{Step size}} \rfloor = \lfloor \frac{\pi/2}{\theta} \rfloor$. Solving $(2k^*+1)\theta \approx \pi/2$ for large N (where $\theta \approx 1/\sqrt{N}$):

$$k^* \approx \frac{\pi}{4}\sqrt{N}$$

This is the origin of the $\sqrt{N}$ scaling. Each iteration rotates by a tiny angle $\sim 1/\sqrt{N}$, and we need $\sim \sqrt{N}$ such rotations to traverse a quarter-circle.

Exercise 8: Optimal Iterations for Different Database Sizes

Calculate k^* for database sizes $N = 4, 16, 64, 256$ and the associated computational speedup over classical search.

Exercise 9: Success Probability After k Iterations

The success probability after k Grover iterations is $P(k) = \sin^2((2k+1)\theta_0)$, where $\theta_0 = \arcsin(1/\sqrt{N})$. For $N = 16$, calculate the success probability after $k = 1, 2, 3, 4$ iterations. Which value of k gives the highest probability?

Concrete Example: 20 qubits

For a database of $N = 2^{20} \approx 10^6$ items:

$$k^* = \frac{\pi}{4}\sqrt{2^{20}} \approx 800 \text{ iterations}$$

Compared to the classical $N/2 \approx 500{,}000$ queries, this is a substantial improvement.

Don't Over-Rotate!

Unlike classical algorithms where more iterations can lead to better results, Grover's algorithm has a "sweet spot." If you iterate beyond k^*, the state rotates *past* $|w\rangle$ and the success probability decreases. You must measure at the right moment.

2.7 Generalization: Multiple Winners

We briefly mention the case where multiple states are considered as winners. The algorithm extends naturally to the case where there are m solutions rather than just one. We redefine the "winner state" as the uniform superposition over all solutions:

$$|w\rangle = \frac{1}{\sqrt{m}} \sum_{x \in \text{Solutions}} |x\rangle$$

The geometric picture remains the same, but now the initial angle is larger: $\sin\theta = \sqrt{m/N}$ instead of $1/\sqrt{N}$. Since we start closer to the target subspace, fewer iterations are needed:

$$k^* = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{m}} \right\rfloor$$

This makes sense: if half the items are winners ($m = N/2$), we only need about one iteration.

Exercise 10: Multiple Winners Initial Angle

For a 4-qubit system ($N = 16$) with $m = 4$ winner states, calculate the initial angle θ and the optimal number of iterations k^*. Compare this to the single-winner case ($m = 1$).

Exercise 11: Multi-Winner Search Angle

Consider a 3-qubit system ($N = 8$) with two winner states: $w \in \{|101\rangle, |110\rangle\}$.

1. Write down the initial state $|s\rangle$ after applying Hadamards.
2. Design an oracle that marks both winning states (hint: it should flip the phase of $|101\rangle$ and $|110\rangle$).
3. Calculate the initial angle θ_0 and determine the optimal number of iterations.
4. Draw the quantum circuit.

2.8 Understanding the Amplitude Dynamics: Inversion About the Mean

The geometric picture is clear, but it's also helpful to see how the amplitudes evolve algebraically. This reveals the mechanism of "inversion about the mean" that gives the diffusion operator its amplifying power. The final state after one iteration is:

$$|\psi_{\text{final}}\rangle = \underbrace{(2|s\rangle\langle s| - I)}_{U_s} \underbrace{(I - 2|w\rangle\langle w|)}_{U_w} \underbrace{H^{\otimes n}|0\rangle^{\otimes n}}_{|s\rangle}$$

After applying the oracle, the state becomes $|s_w\rangle = \frac{1}{\sqrt{N}} \sum_x (-1)^{f(x)}|x\rangle$, where $f(x) = 1$ only for the winner. Let's compute what happens when we apply the diffusion operator $U_s = 2|s\rangle\langle s| - I$:

$$|\psi_{\text{final}}\rangle = 2|s\rangle\langle s|s_w\rangle - |s_w\rangle$$

. Using the fact that $\langle s|x\rangle = \frac{1}{\sqrt{N}}$, the full expression for the amplitude becomes:

$$|\psi_{\text{final}}\rangle = \frac{1}{\sqrt{N}} \sum_x \left[\frac{2}{N} \underbrace{\sum_z (-1)^{f(z)}}_{\text{Sum over all phases}} - (-1)^{f(x)} \right] |x\rangle$$

The key observation is that $\langle s|s_w\rangle = \frac{1}{N}\sum_x (-1)^{f(x)} = \frac{N-2}{N}$ (since $N-1$ terms contribute $+1$ and one term contributes -1). After some algebra, the amplitude of state $|x\rangle$ becomes:

$$\frac{2(N-2)}{N} - (-1)^{f(x)} \cdot 1$$

State	$f(x)$	$(-1)^{f(x)}$	New Amplitude	Effect
Loser	0	+1	$\frac{2}{N}(N-2) - 1 = 1 - \frac{4}{N}$	From 1 → **Decrease**
Winner	1	-1	$\frac{2}{N}(N-2) - (-1) = 3 - \frac{4}{N}$	From 1 → **Roughly** $\times 3$

The winner's amplitude roughly triples with each iteration (for large N), while loser amplitudes decrease slightly. This is the quantitative manifestation of "inversion about the mean": the winner amplitude, being far below the mean, gets reflected far above it.

2.9 The Physical Origin of Interference

To finish discussing the algorithm and before switching to examples and experimental realisations, let's wonder where the quantum interference takes place. You remember from previous lecture that a quantum computer's advantage does not rely in running the same calculation in parallel over all the superposed states. Rather, you only want a single state in your final state distribution. The speedup in Grover's algorithm comes from this quantum interference, and it's worth understanding where this interference arises:

- **The Oracle Step:**
  - By flipping the winner's phase to negative, the oracle creates a situation where the winner amplitude is now "out of step" with all the others.
  - The mean amplitude drops slightly because of this negative contribution.
- **The Diffusion Step:** The diffusion operator reflects all amplitudes about this new (lower) mean.
  - For the loser states, which sit just above the mean, reflection barely moves them.
  - But the winner state sits far below the mean, so reflection pushes it upward, effectively collecting constructive contributions from all the other states, the $\times_3$ factor we saw in the previous section.

This is **constructive interference** for the winner and **destructive interference** for the losers, achieved not by path differences (as in optical interference) but by manipulation of quantum phases.

Intuition on the Origin of Speedup: Pythagoras

Where does the speedup come from? You can see it from **Pythagoras**' theorem. In the classical world, different observable states are like pure coordinate directions (the axes of our Hilbert space). To get from your starting point to the correct answer, you are essentially "walking the edges" of an N-dimensional space.

- **Classical (Manhattan distance):** You have to traverse N items one by one. You are constrained to move along the axes (checking one bit string at a time). To reach the "opposite corner" of a unit cube in N dimensions, you would have to walk N units. Distance $\propto N$.
- **Quantum (Euclidean distance):** Quantum mechanics gives us access to **diagonal directions**

(superpositions) that are unavailable to a classical system.
  - By allowing the state vector to move "diagonally" through the Hilbert space, we can take a shortcut.
  - In an N-dimensional space, the distance from the origin to the opposite corner $(1, 1, \ldots, 1)$ is $\sqrt{1^2 + 1^2 + \cdots + 1^2} = \sqrt{N}$.
  - The Grover iteration effectively traces a quarter-circle arc from our initial state to the target, utilizing these "diagonal" states to achieve a square-root shortcut.
  - The state vector slowly walks along this arc, taking a path that would be entirely unavailable if we were limited to pure coordinate directions.

A word of caution: the "Quadratic Trap"

It is proven (see BBBV Theorem) that $\mathcal{O}(\sqrt{N})$ is the lower bound: we cannot search faster than square root. When we will discuss resource estimation and space-time overhead (how many qubits and how much time) of applications, we will see that the $\mathcal{O}(\sqrt{N})$ for quantum vs $\mathcal{O}(N)$ for classical complexity will not be enough: we need to take the prefactors into account if we ever want to have a quantum advantage over classical computers, eg $C \cdot N$ vs $Q \cdot \sqrt{N}$. The "Quadratic Trap" occurs when the quantum computer's clock speed is slower than the classical computer. If the "Quantum constant" Q (due to error correction overheads that we will cover in the next lectures) is 10^8 times larger than the "Classical constant" C, then for small N, the classical computer remains much faster. Said in another way:

$$C \cdot N < Q \cdot \sqrt{N} \implies \sqrt{N} < Q/C$$

If $Q/C = 10^8$, Grover only wins if the database size $N > 10^{16}$.

2.10 Example: 2-Qubit Search ($N = 4$)

Let's work through Grover's algorithm for the smallest non-trivial case: searching among 4 items with 2 qubits. In this example, we'll suppose the winner is $|w\rangle = |11\rangle$.

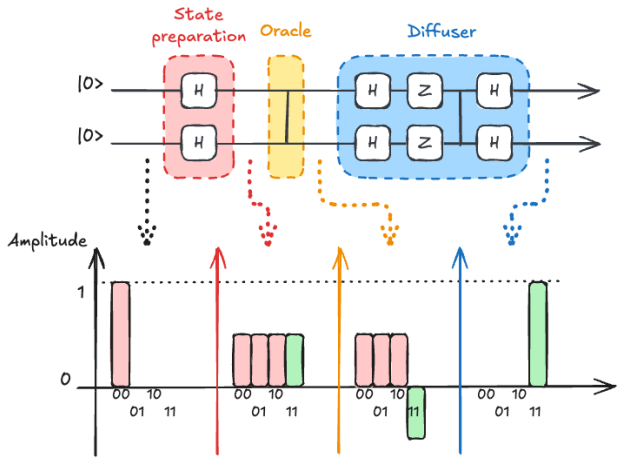


Figure 4: Detailed circuit implementation of Grover's algorithm with gate decomposition

2.10.1 Initial Setup

The starting angle is $\theta_0 = \arcsin(1/\sqrt{4}) = \arcsin(1/2) = \pi/6$ (30 degrees). Since we need to rotate from $\pi/6$ to $\pi/2$, the optimal number of iterations is $k^* = 1$. This is a special case where a single Grover iteration gives perfect success probability.

2.10.2 The Oracle

For winner state $|11\rangle$, the oracle flips only this state's phase:

$$U_w|s\rangle = U_w \cdot \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$

In matrix form, this oracle is simply the controlled-Z gate: $U_w = CZ$, seen in lecture 1:

$$U_w = \begin{pmatrix} 1 & & & \\ & 1 & (0) & \\ & (0) & 1 & \\ & & & -1 \end{pmatrix} = CZ$$

2.10.3 Implementing the Diffusion Operator

The diffusion operator $U_s = 2|s\rangle\langle s| - I$ might look complicated, but there's a clear circuit decomposition. The key insight is that reflecting about $|s\rangle$ is equivalent to:

1. Rotating $|s\rangle$ to $|0\rangle$: $H^{\otimes n}$
2. Reflecting about $|0\rangle$: U_0
3. Rotating back: $H^{\otimes n}$

Where U_0 flips the sign of $|00...0\rangle$ and leaves all other states unchanged. The operator U_0 can be implemented as a **multi-controlled Z gate** (flip sign when all control qubits are 0). For 2 qubits, this decomposes to $U_0 = Z_1 Z_2 \cdot CZ$:

$$U_0 \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle - |11\rangle)$$

The complete circuit has three stages: **Initialization** (Hadamards to create $|s\rangle$), **Oracle** (CZ gate marking the winner), and **Diffuser** (H-Z-CZ-H sequence), see fig. 4.

A more complex example with 5 qubits and one winner can be seen using Quirk.

2.11 Qiskit Examples

Let's look at examples that you can run yourself.

2.11.1 Examples with 2 And 3 Qubits

First go to Colab, a colaborative jupyter notebook website. Then click on "Open Notebook" for a new Colab notebook and paste the Github url from a Grover implementation edited from the official qiskit release: qiskit-demo/presentations/grover.ipynb at main · francois-marie/qiskit-demo.

2.11.2 Application: Sudoku

How would you encode a 2×2 Sudoku puzzle as a Grover search problem? Let's do this using Qiksit and Colab!

2.12 Experimental Implementations

We end this first part of the lecture by looking at some experimental implementations. Grover's algorithm has been demonstrated on many quantum computing platforms: historically NMR but also superconducting (SC), trapped ions, photonics, and spins. The table below summarizes key experimental results, showing the gap between theoretical and achieved success probabilities, a measure of hardware quality.

Platform	Reference	Qubits	Database items	Winners	Exp. Success Rate	Theo. Success Rate
SC	[4]	3	8	-	-	
SC	[5]	6	64	-	4%	
SC	[5]	4	32	-	45%	
Trapped Ions	[6]	3	8	1	0.39	0.78
Trapped Ions	[6]	3	8	2	0.70	1
Photonics	[7]	2	4	1	0.71	
Silicon Spins	[8]	3	8	1	0.93	0.98
Silicon Spins	[8]	3	8	2	0.96	1
NMR	[2]	2	4	1	-	

Notice that silicon spin qubits achieve high fidelities, approaching theoretical limits. The gap between achieved and theoretical success rates reflects gate errors and decoherence in current hardware.

3 Quantum Fourier Transform and Phase Estimation

Having explored Grover's quadratic speedup for unstructured search, we now turn back to last week's lecture, to a different class of algorithms that achieve exponential advantages. The Quantum Fourier Transform and Quantum Phase Estimation are the workhorses behind Shor's factoring algorithm but also quantum chemistry simulations, and many other applications. While Grover taught us about amplitude amplification through geometric rotations, QFT and QPE reveal how quantum computers can extract hidden periodicities and eigenvalues exponentially faster than classical approaches.

3.1 Classical Fourier Transform

Let's start by recalling some intuition on the Fourier transform. The Fourier transform is one of the most powerful tools in mathematics and engineering. It converts signals from the time domain to the frequency domain, revealing hidden periodicities, see fig. 5 that illustrates the concept of Fourier transformation, adapted from Wikipédia's page on the Fourier transform. A time-domain signal f containing multiple frequency components (in black, top row) is decomposed into its constituent frequencies (red, green and blue, middle row). The Fourier transform reveals the amplitude and phase of each frequency component (bottom row), making hidden periodicities visible. The key insight is that periodic structure in the time domain becomes localized peaks in the frequency domain. Mathematically, the transform $\hat{f}(\xi)$ tells us "how much" of each frequency ξ is present in the original signal:

$$\hat{f}(\xi) = \int_{-\infty}^{+\infty} f(x) e^{-2\pi i \xi x} dx$$

The quantum version performs this transformation on quantum amplitudes rather than classical signals but the idea remains the same.

Exercise 12: Comparing Classical and Quantum Fourier Transform Complexity

The classical Fast Fourier Transform (FFT) on $N = 2^n$ points requires $O(N \log N)$ operations. The Quantum Fourier Transform on n qubits requires $O(n^2)$ gates. Compare these complexities for $n = 10, 20, 30$ qubits in terms of number of operations and compute the speedup factor.

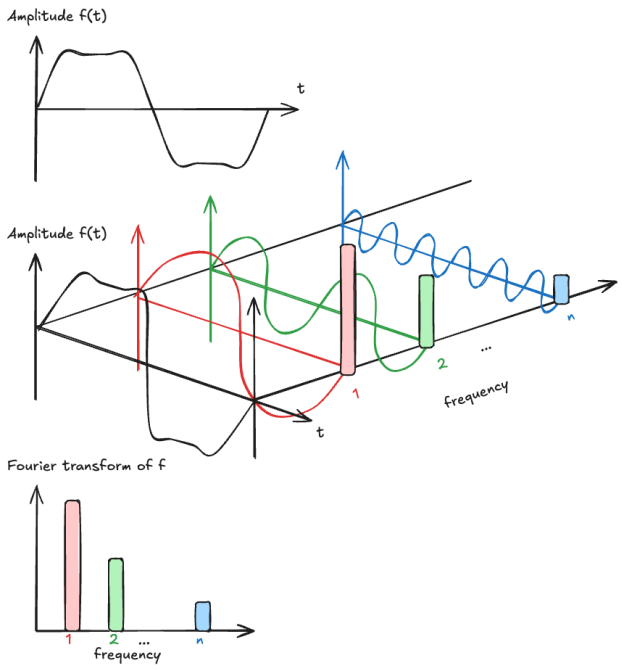


Figure 5: The Fourier transform relates the time domain, in black, with a function in the domain of the frequency. The component frequencies are shown as red green and blue peaks in the domain of the frequency.

3.2 Discrete Fourier Transform (DFT)

For computational purposes, we can work with discrete vectors rather than continuous functions. Given an input vector $(x_0, x_1, \ldots, x_{N-1})$, the DFT produces an output vector $(y_0, y_1, \ldots, y_{N-1})$ where:

$$y_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \, \omega_N^{nk}, \quad \text{where } \omega_N = e^{2\pi i/N}$$

The quantity ω_N is the primitive Nth root of unity, a complex number that, when raised to the Nth power, equals 1.

3.3 The Quantum Fourier Transform

Going from DFT to QFT is simply switching from vectors to vector states: the QFT performs the DFT on the amplitudes of a quantum state. If we start with a state $|\psi\rangle = \sum_{j=0}^{N-1} x_j |j\rangle$, the QFT produces:

$$QFT|\psi\rangle = \sum_{k=0}^{N-1} y_k |k\rangle, \quad \text{where } y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \, \omega_N^{jk}$$

For a computational basis state $|j\rangle$, this becomes:

$$QFT|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} |k\rangle$$

The QFT creates a superposition where each basis state $|k\rangle$ acquires a phase that depends on both j (the input) and k (the output index).

3.4 Matrix Representation

Mathematically, the QFT can be expressed as an $N \times N$ unitary matrix:

$$F_N = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)^2} \end{pmatrix}$$

Each row represents a different "frequency", and the entries oscillate faster as you move down the matrix.

Exercise 13: Construct F4 Matrix

For $N = 4$, we have $\omega = e^{i\pi/2} = i$. Write out the explicit 4×4 QFT matrix using powers of i. Verify that the first column corresponds to $QFT|0\rangle$.

3.5 Key Properties

For the sake of completeness, we briefly mention some key properties of the QFT. The QFT is a **unitary** transformation: $F_N F_N^\dagger = I$. This means it preserves probability (as all quantum operations must) and is reversible. The inverse QFT is simply $F_N^\dagger$, obtained by taking the complex conjugate of all entries.

Exercise 14: Roots of Unity Properties

Let $\omega_8 = e^{2\pi i/8}$ be the primitive 8th root of unity. Calculate ω_8^4 and ω_8^8. Show that $\sum_{k=0}^{7} \omega_8^{jk} = 0$ for $j \neq 0 \pmod 8$.

3.6 Building Intuition: How the QFT Encodes Information

Before diving into the circuit representation of the QFT, we build some intuition by looking at an example of QFT on basis states. A feature of the QFT is that when applied to a basis state $|j\rangle$ (eg $|j\rangle = |5\rangle = |101\rangle$), the output can be written as a tensor product of single-qubit states. As we will see explicitly by taking the example of 3 and 4 qubits, each output qubit is in a superposition $|0\rangle + e^{i\phi_k}|1\rangle$, where the phase ϕ_k depends on the input j:

$$QFT_N|j\rangle = \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} \left(|0\rangle + e^{2\pi i j/2^k}|1\rangle\right)$$

In binary notation, where $j = j_1 j_2 \cdots j_n$ and $0.j_j j_{j+1} \cdots j_n$ denotes the binary fraction, this becomes:

$$QFT_N|j\rangle = \frac{1}{\sqrt{N}} \left(|0\rangle + e^{2\pi i[0.j_n]}|1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i[0.j_{n-1}j_n]}|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{2\pi i[0.j_1 \cdots j_n]}|1\rangle\right)$$

The different qubits rotate at different "frequencies":

- The least significant output qubit rotates by π between consecutive inputs (fastest oscillation) hence oscillates between $|+\rangle$ and $|-\rangle$,
- The next qubit rotates by $\pi/2$ (half the frequency), hence oscillates between $|+\rangle, |i\rangle, |-\rangle, |-i\rangle$
- Each subsequent qubit rotates half as fast

This hierarchical structure of phases is what makes the QFT useful for detecting periodicity (on what relies Shor's algorithm): if the input state has period r, the output will have peaks at multiples of N/r.

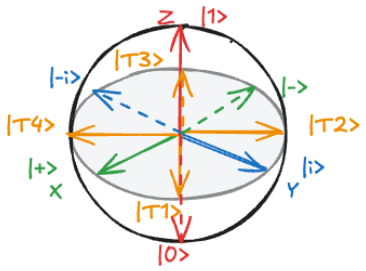


Figure 6: Bloch sphere representation of QFT output states showing phase relationships

As shown in fig. 6, the QFT output states can be visualized on the Bloch sphere with their phase relationships. Let's map explicitly input number states to their binary representations and QFT outputs in the following table:

Input state	Binary $\|q_1q_2q_3\rangle$	QFT Output $\|\tilde{q}_1, \tilde{q}_2, \tilde{q}_3\rangle$
$\|0\rangle$	$\|000\rangle$	$\|+,+,+\rangle$
$\|1\rangle$	$\|001\rangle$	$\|-,i,T_1\rangle$
$\|2\rangle$	$\|010\rangle$	$\|+,-,i\rangle$
$\|3\rangle$	$\|011\rangle$	$\|-,-i,T_2\rangle$
$\|4\rangle$	$\|100\rangle$	$\|+,+,-\rangle$
$\|5\rangle$	$\|101\rangle$	$\|-,i,T_3\rangle$
$\|6\rangle$	$\|110\rangle$	$\|+,-,-i\rangle$
$\|7\rangle$	$\|111\rangle$	$\|-,-i,T_4\rangle$

The first qubit oscillates from $|+\rangle, |T_1\rangle, \ldots, |-i\rangle, |T_4\rangle$.

The pattern reveals how the input number j gets encoded into the relative phases of the output qubits.

Worked Example: QFT of state 5

For the 3-qubit state $|5\rangle = |101\rangle$, the QFT produces phases that increase with qubit position:

- The fastest qubit ($\tilde{q}_3$) rotates by $5 \times \frac{2\pi}{8} = \frac{5\pi}{4}$
- The middle qubit ($\tilde{q}_2$) rotates twice as slowly: $\frac{5\pi}{2} \mod 2\pi = \frac{\pi}{2}$
- The slowest qubit ($\tilde{q}_1$) rotates at half that rate: $5\pi \mod 2\pi = \pi$

Let's write the full explicit tensor product form for QFT on $|5\rangle$:

$$QFT|5\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi/2}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{i5\pi/4}|1\rangle)$$

Exercise 15: Product State for Specific Input

Calculate the explicit product state representation for $QFT|3\rangle$ on a 3-qubit system ($N = 8$). Express your answer as a tensor product of single-qubit states $\frac{1}{\sqrt{2}}(|0\rangle + e^{i\phi_k}|1\rangle)$.

Now that we built some intuition on what the QFT does in practice, let's look at its circuit implementation.

3.7 QFT Circuit Implementation

The feature of the QFT is that it can be implemented with only $O(n^2)$ gates (even $O(n \log n)$ using approximate techniques) for n qubits (where $N = 2^n$), compared to the $O(N \log N)$ operations of the classical FFT. The circuit uses two types of gates:

Hadamard gate: Creates equal superposition of $|0\rangle$ and $|1\rangle$:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Controlled phase rotation: Adds a phase depending on the control qubit:

$$\text{UROT}_k = R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}$$

The circuit of fig. 7 illustrates the complete circuit structure and applies Hadamards and controlled rotations in a specific pattern, followed by SWAP gates to reverse the qubit order. The algorithm proceeds as follows:

1. apply a Hadamard to the first qubit, then controlled-R_2 (controlled by qubit 2), controlled-R_3 (controlled by qubit 3), and so on.
2. Then move to the second qubit and repeat with smaller rotations.

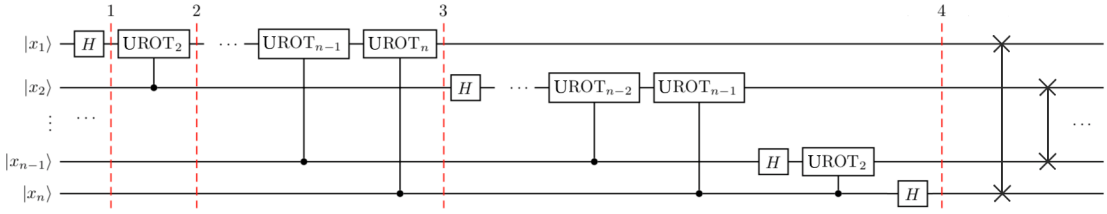


Figure 7: QFT circuit showing Hadamard gates and controlled rotation gates

Exercise 16: QFT Circuit Depth

For an n-qubit QFT, count the total number of gates required. How many Hadamard gates, controlled rotation gates, and SWAP gates are needed? What is the total circuit depth (assuming all gates are sequential)?

A nice and interactive display of the QFT circuit on 8 qubits can be found on Quirk.

3.8 The Inverse QFT

Since the QFT is unitary, we briefly mention that its inverse $QFT^\dagger$ exists and undoes the transformation. In terms of circuit implementation, the inverse QFT circuit is simply the original circuit run backwards: reverse the order of all gates and replace each R_k with $R_k^\dagger = R_k^{-1}$ (which has the opposite phase rotation). The inverse QFT is important for extracting classical information from quantum phase information, we now focus on this in action in Quantum Phase Estimation use case.

Exercise 17: Inverse QFT Circuit

Describe the circuit for the inverse QFT on 3 qubits. What are the gate types, and in what order do they appear? How does it differ from the forward QFT circuit?

3.9 Do it Yourself!

Just like for Grover's algorithm, let's look at examples that you can run yourself. Go again to Colab, a colaborative jupyter notebook website. Then click on "Open Notebook" for a new Colab notebook and paste the following Github url: qiskit-demo/presentations/quantum-fourier-transform.ipynb at main · francois-marie/qiskit-demo.

3.10 Application: Quantum Phase Estimation (QPE)

The QFT finds one of its most important application in Quantum Phase Estimation, a subroutine used by Shor's factoring algorithm and many quantum chemistry algorithms. Let's start by quickly explaining the problem at stake.

3.10.1 The Problem

You are given a unitary operator U and one of its eigenvectors $|v\rangle$ satisfying $U|v\rangle = e^{2\pi i\theta}|v\rangle$, the goal is to estimate the eigenvalue phase θ. Looks pretty straightforward, right? But how would you do this in practice?

3.10.2 The Key Insight: the return of the Phase Kickback

QPE uses controlled-U operations to transfer the phase information from the eigenvector into an auxiliary register, just like what we used in Grover's Algorithm! If we apply $C\text{-}U^{2^j}$ with the control qubit in superposition and the target in eigenstate $|v\rangle$:

$$C\text{-}U^{2^j}\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |v\rangle\right) = \frac{|0\rangle + e^{2\pi i \cdot 2^j\theta}|1\rangle}{\sqrt{2}} \otimes |v\rangle$$

We see that by using multiple control qubits with different powers of U, we encode θ into the phases of a multi-qubit register, the format that the inverse QFT can decode. The circuit implementation is shown in fig. 8:

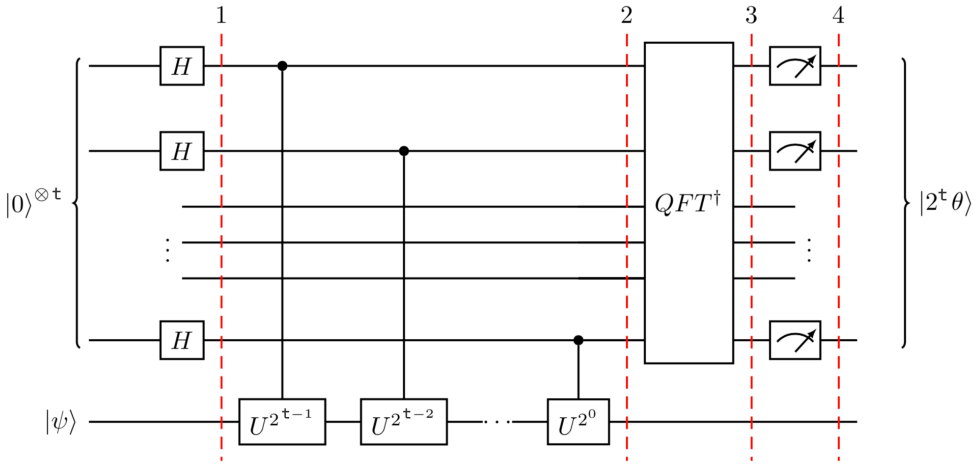


Figure 8: Quantum Phase Estimation circuit with controlled-U gates and inverse QFT

Exercise 18: Controlled-U Power Implementation

Suppose U is a unitary with eigenstate $|v\rangle$ and eigenvalue $e^{2\pi i\theta}$ where $\theta = 3/16$. Calculate the phase acquired by the control qubit when applying $C\text{-}U^{2^2}$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |v\rangle$. Express your answer as $e^{i\phi}$ and determine ϕ.

The depicted circuit above can be hard to understand so to make things hopefully clearer, let's take an example. Consider the T gate (also called $\pi/8$ gate) that we have lentoined several times already throughout the previous lectures:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

The state $|1\rangle$ is an eigenvector of T with eigenvalue $e^{i\pi/4} = e^{2\pi i \cdot (1/8)}$. Thus $\theta = 1/8$ is the value we are going to be looking for. If we run QPE with 3 qubits in the auxiliary register, the algorithm will encode $\theta = 1/8 = 0.001_2$ in binary. After the inverse QFT, we measure the auxiliary register and obtain the binary representation of θ:

$$\theta = \frac{\text{measured value}}{2^n} = \frac{1}{8}$$

This example illustrates the power of QPE: it converts quantum phase information, which is normally invisible to measurement, into a classical bitstring that we can read out directly.

Exercise 19: QPE for the T Gate

What would you measure if you used only 2 qubits in the auxiliary register instead of 3? Would the result still be exact?

Exercise 20: Phase Estimation Accuracy

In QPE, the auxiliary register has n qubits, allowing us to estimate phases to n bits of precision. If we want to estimate a phase θ with accuracy ϵ (i.e., within $\pm\epsilon$), how many auxiliary qubits do we need? What is the probability of success if we use exactly this many qubits?

3.10.3 Qiskit time

We end this section by running some code. Just like for Grover's algorithm and QFT, let's look at examples that you can run yourself. Go again to Colab, a colaborative jupyter notebook website. Then click on "Open Notebook" for a new Colab notebook and paste the following Github url: qiskit-demo/presentations/quantum-phase-estimation.ipynb at main · francois-marie/qiskit-demo.

4 Resource Estimates

Between last week's lecture and today's, we've now seen two paradigms of quantum advantage: Grover's quadratic speedup for search and the exponential speedup of Shor, QFT and phase estimation. But there's a crucial question we haven't addressed yet. Can we actually run these algorithms on real hardware? How many would would we need and how much time would it take to run a *useful* quantum application? Answering these questions provides what's called **space-time overheads** and is crucial to understand the cost of quantum advantage. The answer depends not just on the number of logical (aka perfect and noiseless) qubits required, but on error rates and coherence times of physical qubits, and the massive overhead of quantum error correction that we will discuss in the next two lectures. Let's end this lecture by confronting the resource requirements and understand the gap between algorithmic promise and experimental reality.

4.1 (Partial) Landscape of Quantum Algorithms

We've said already that different problem classes admit different types of quantum speedups, see fig. 9:

- Problems like factoring, discrete logarithm, or HHL algorithm where quantum computers achieve exponential advantage (in blue),
- Optimization and search problems where we see quadratic speedups (yellow),
- Other types of problems, like finding the ground state of an Hamiltonian or combinatorial problems, with an unproven advantage (red)

To classify all these quantum applications, we first decompose them in terms of the algorithm they use (for instance, remember that Grover's algorithm is at the center of a wide variety of use cases) and the *primitives* or building blocks of the algorithm. More detailed maps exist but they become quickly unreadable. However, considering even such a partial landscape allows to take a step back on the many topics addressed during our two "Application" lectures and all of the many remaining ones that you will discover in your future careers.

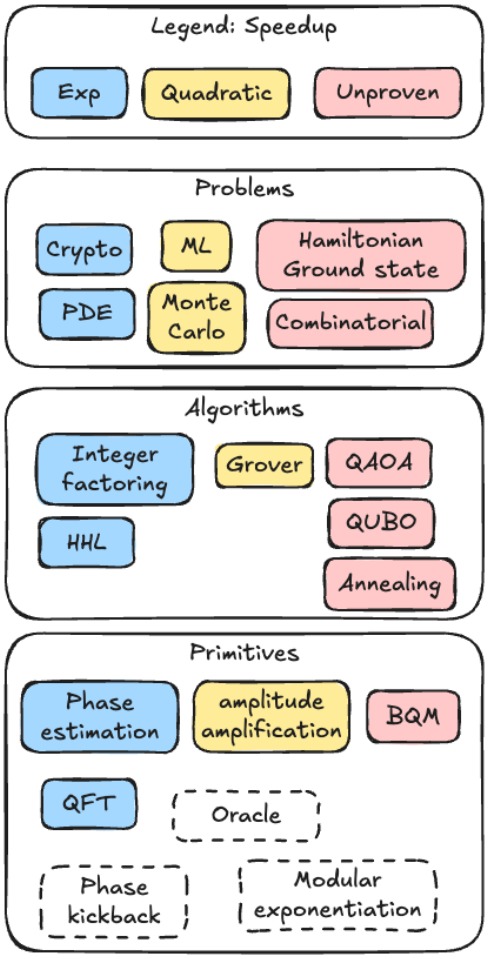


Figure 9: Partial landscape of quantum algorithms showing primitives, algorithms, problem classes and speedup types

4.2 Resource Overheads: From Theory to Reality

Let's talk numbers. When researchers publish quantum algorithms, they often focus on asymptotic complexity (the $\mathcal{O}(...)$ notation that tells you how runtime scales). But if you want to actually run these algorithms on hardware one day, you need concrete resource estimates beforehand. How many qubits? How many gates? How long will it take? What error rates can we tolerate?

Take nitrogen fixation as an example. For context, the FeMoCo (iron-molybdenum cofactor) is the active site in certain bacteria that can convert atmospheric nitrogen into ammonia at room temperature. If we could

understand this process well enough to replicate it industrially, we could revolutionize agriculture and reduce the massive energy costs of the Haber-Bosch process. Quantum simulation of FeMoCo is often cited as a "killer app" for quantum computers because the electronic structure involves strongly correlated electrons that are extremely difficult for classical computers to handle accurately.

But here's the reality check. Even for this relatively small molecule (Fe_7MoS_9C), you need somewhere between 100 to 1,000 logical qubits and around 10^{14} (!!) logical gates. Besides, that's not 100-1000 physical qubits (that we can build that today), but *error-corrected* logical qubits, each potentially requiring thousands of physical qubits depending on your error correction code and the quality of your hardware (more on that next week). The tradeoff between circuit depth (number of sequential gates) and width (number of qubits) depends on the specific problem and algorithm. Financial optimization problems can require even more qubits (around 10,000 logical qubits) but fewer total gates.

For Shor's algorithm attacking RSA-2048 (the encryption standard used widely on the internet), you need about 1,000 logical qubits. But when you account for quantum error correction, that balloons to roughly 1 million physical qubits for the Gidney estimate [9], or 20 million for more conservative assumptions or other platforms [10]. And you would need to run the computation for about a week, maintaining coherence and performing error correction continuously throughout.

App.	**FeMoCo (Chemistry)** [11]	**Financial opti** [12]	**RSA-2048 (Shor, SC)** [9]	**RSA-2048 (Shor, NA)** [10]
Logical Qubits	$\sim 100 - 1000$	$\approx 10^4$	$\sim 1,000$	$\sim 10,000$
Logical Gates	$\sim 10^{14}$	$\sim 10^{10}$	$\sim 10^{10}$	$\sim 10^{10}$ (CCZ)
Physical Qubits	-	-	~ 1 Million	~ 20 Million
Runtime	Days to months		~1 week	~1 week

4.3 Bridging Theory and Practice

The gap between what we can build and what we need is real, but it's closing. See fig. 10 from Quantum Landscape which shows historical progress in reducing resource estimates for problems related to cryptography. Back in the early 2000s, estimates for factoring RSA-2048 were in the billions of qubits. Today, through better algorithms and more efficient error correction schemes, we're down to the millions. That's still far from the hundreds or thousands of qubits in current machines, but the trajectory is clear. Will this be sustainable or will it plateau? Still an open question.

4.4 Where We Stand Today

Also from Quantum Landscape, fig. 11 provides a snapshot of where we are right now. Current quantum processors have somewhere between 100 and 1,000 physical qubits, depending on the platform. Superconducting qubits (IBM, Google, Rigetti) are pushing toward the high hundreds. Trapped ions (IonQ, Quantinuum) have fewer qubits but better connectivity and gate fidelities. Neutral atoms (QuEra, Pasqal) can scale to larger numbers but are still working on gate fidelities.

The vertical axis in this figure is linked to the number of logical operations you can perform reliably. Right now, we're in the NISQ (Noisy Intermediate-Scale Quantum) era, meaning we have devices with somehow a decent number of qubits but not enough coherence or low enough error rates to implement reach what is needed for useful algorithm, on the order of 10^{-10}. We can run short circuits (maybe a few hundred to a thousand gates) before errors accumulate and destroy the computation.

But look at where the useful algorithms sit. Shor's algorithm needs millions of gates. Quantum chemistry simulations need even more. The applications that would provide clear practical advantage are still out of reach compared to today's achieved performance.

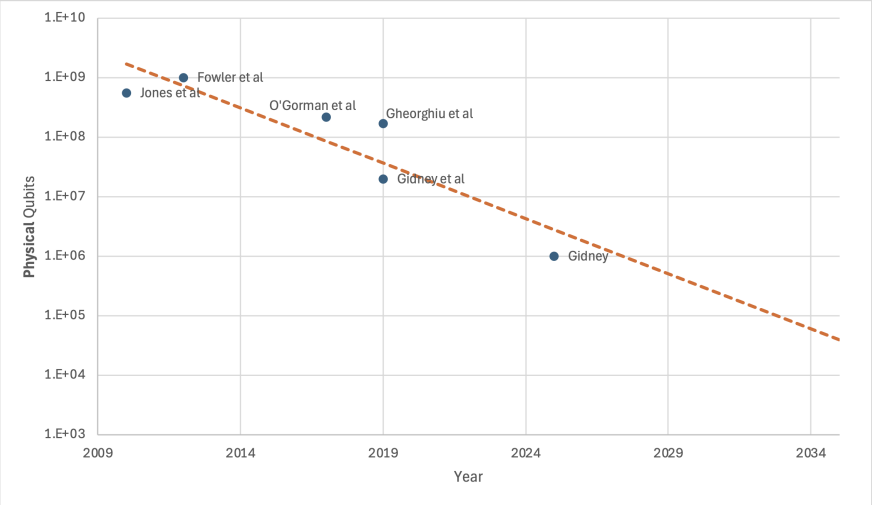


Figure 10: Historical progress in reducing resource estimates for cryptographically-relevant quantum algorithms

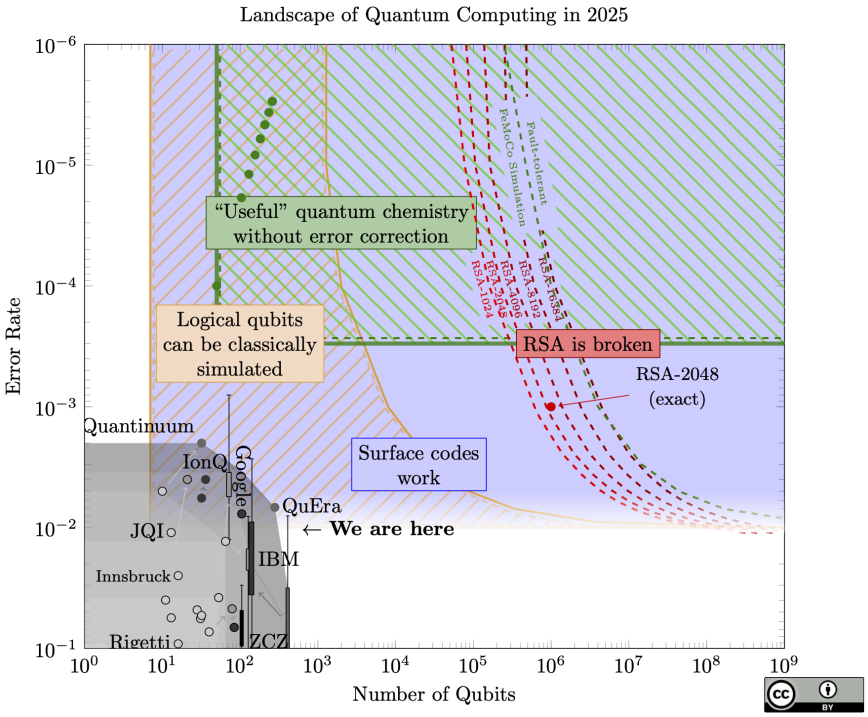


Figure 11: Current quantum computing landscape showing the gap between available hardware and algorithm requirements

Even if we can increase the number of qubits, useful computations would still be out of reach because of the error rates gap between what we have and what is required. This is precisely why quantum error correction is the next major topic in this course: building one almost-noise-free qubit out of many noisy ones. QEC is the bridge the gap between NISQ devices and fault-tolerant quantum computers that can run Shor, simulate molecules, and tackle the problems we've been discussing. Without error correction, we're limited to small demonstrations and variational algorithms that try to squeeze utility out of noisy circuits. With error correction, we unlock the full algorithmic power of quantum computing.

4.5 The Path Forward

The takeaway of this section isn't pessimism, it's realism. Quantum algorithms provide genuine speedups (quadratic for Grover, exponential for Shor and quantum simulation). But translating those asymptotic advantages into practical utility requires overcoming significant engineering challenges. We need:

- **Better qubits:** Lower error rates, longer coherence times
- **More qubits:** Scaling from hundreds to millions
- **Error correction:** Efficient codes that don't require prohibitive overhead
- **Compilation and optimization:** Smarter ways to map algorithms onto hardware constraints

The next lectures on quantum error correction will show you exactly how we plan to get there. We'll see how the surface code and other QEC schemes can protect quantum information from noise, how to perform fault-tolerant gates on encoded qubits (aka robust to physical error propagating), and what it will take to reach the thresholds where error-corrected quantum computing becomes viable. The resource estimates we've discussed today aren't fixed targets but they're moving goals that improve and get closer to reality as we develop better codes, better compilers, and better hardware. But they give you a realistic picture of where the field stands and what needs to happen next.

5 Conclusion

In this lecture, we've explored three pillars of the quantum algorithm landscape. Grover's algorithm showed us how quantum interference and geometric rotations in Hilbert space can achieve a quadratic speedup for unstructured search, with the insight that two reflections compose into a rotation. The Quantum Fourier Transform revealed how quantum computers can extract periodicities exponentially faster than classical FFTs, by encoding information into relative phases of product states. And Quantum Phase Estimation demonstrated how phase kickback and the inverse QFT can convert hidden eigenvalue information into measurable classical bits, powering applications from factoring to quantum chemistry.

But we also confronted a sobering reality. The gap between theoretical speedups and practical implementations remains vast. Shor's algorithm can break RSA-2048 in polynomial time, but doing so requires millions of physical qubits and weeks of coherent operation. This gap isn't a fundamental limitation, it's an engineering challenge.

In the upcoming lectures, we'll see how quantum error correction works to bridge this gap, starting with simple repetition codes and building up to the surface code and other codes that are the leading candidates for fault-tolerant quantum computing. We'll understand the threshold theorem that proves error correction can work even with imperfect gates, and we'll see what it takes to perform logical operations on encoded qubits without destroying the error protection.

References

1. Chen, Z. *et al.* Exponential suppression of bit or phase errors with cyclic error correction. *Nature* **595**, 383–387 (2021).
2. Chuang, I. L., Gershenfeld, N. & Kubinec, M. Experimental Implementation of Fast Quantum Searching. *Phys. Rev. Lett.* **80**, 3408–3411 (1998).

3. Grover, L. K. A fast quantum mechanical algorithm for database search. in *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing* 212–219 (Association for Computing Machinery, New York, NY, USA, 1996). doi:10.1145/237814.237866.
4. AbuGhanem, M. Characterizing Grover search algorithm on large-scale superconducting quantum computers. *Sci Rep* **15**, 1281 (2025).
5. Chu, J. *et al.* Scalable algorithm simplification using quantum AND logic. *Nat. Phys.* **19**, 126–131 (2023).
6. Figgatt, C. *et al.* Complete 3-Qubit Grover search on a programmable quantum computer. *Nat Commun* **8**, 1918 (2017).
7. Main, D. *et al.* Distributed quantum computing across an optical network link. *Nature* **638**, 383–388 (2025).
8. Thorvaldson, I. *et al.* Grover's algorithm in a four-qubit silicon processor above the fault-tolerant threshold. *Nat. Nanotechnol.* **20**, 472–477 (2025).
9. Gidney, C. How to factor 2048 bit RSA integers with less than a million noisy qubits. (2025) doi:10.48550/arXiv.2505.15917.
10. Zhou, H. *et al.* Resource Analysis of Low-Overhead Transversal Architectures for Reconfigurable Atom Arrays. (2025) doi:10.48550/arXiv.2505.15907.
11. Reiher, M., Wiebe, N., Svore, K. M., Wecker, D. & Troyer, M. Elucidating Reaction Mechanisms on Quantum Computers. *Proc. Natl. Acad. Sci. U.S.A.* **114**, 7555–7560 (2017).
12. Chakrabarti, S. *et al.* A Threshold for Quantum Advantage in Derivative Pricing. Preprint at https://doi.org/10.48550/arXiv.2012.03819 (2021).