UNIVERSITÀ DEGLI STUDI DI TORINO

DIPARTIMENTO DI INFORMATICA
SCUOLA DI SCIENZE DELLA NATURA

Degree Course in Computer Science (Corso di Laurea in Informatica)



UNIVERSITÀ
DI TORINO

BACHELOR'S DEGREE THESIS IN COMPUTER SCIENCE

Simulation of Quantum Algorithms Using the Q# Language

Original Italian Title: "Simulazione di Algoritmi Quantistici usando lo strumento Q#"

Supervisors:

Prof. Stefano Berardi
Supervisor

Prof. Luca Paolini
Co-supervisor

Candidate:

Daniel Biasiotto
Student ID (Matricola): 914117

Graduation Session: November 2022
Academic Year 2021/2022

TRANSLATION NOTICE & ACADEMIC CERTIFICATION

Notice: This document is an English translation of the original Bachelor's thesis written in Italian and defended in November 2022 at the University of Turin, Department of Computer Science. Translated for academic evaluation purposes.

- **Original Title:** *Simulazione di Algoritmi Quantistici usando lo strumento Q#*
- **Candidate:** Daniel Biasiotto (Student ID: 914117)
- **Candidate Profile:** linkedin.com/in/daniel-biasiotto · github.com/denialbb · denial.in.net
- **Degree Conferred:** Bachelor's Degree in Computer Science (*Laurea di Primo Livello in Informatica*), Final Mark: 110/110 Cum Laude (*Centodieci e Lode*).
- **Institution:** Università degli Studi di Torino, Dipartimento di Informatica, Torino, Italy.

Statement of Originality and Academic Integrity

"I hereby declare that I am responsible for the content of the thesis presented for the purpose of obtaining the degree, that I have not plagiarized in whole or in part the work produced by others, and that I have cited the original sources in accordance with current regulations concerning plagiarism and copyright. I am also aware that should my declaration prove mendacious, I may incur the sanctions provided for by law and my admission to the final examination could be revoked."

Daniel Biasiotto

Turin, November 2022

Contents

1 Introduction 1
1.1 Definition of a Quantum Computer 1
1.2 Hardware Limits 2
1.3 Technological Applications and Computational Complexity 3
2 Environment and Tooling 6
3 The Q# Programming Language 8
4 Quantum Oracles 11
5 The Deutsch-Jozsa Algorithm 15
5.1 The Classical Solution 15
5.2 The Quantum Solution 16
6 Quantum Teleportation 21
6.1 Mathematical Formulation 21
6.2 Implementation in Python and Q# 24
7 Conclusions 26
8 Bibliography 27

1 Introduction

The development of quantum software has emerged over recent decades as a research field of paramount theoretical and technological interest, offering compelling avenues for surmounting the fundamental computational boundaries encountered across diverse scientific domains. In targeted application areas, quantum algorithms possess the proven potential to replace or dramatically outperform their classical counterparts:

- **Cryptography:** Quantum key distribution (QKD) protocols offering information-theoretic security, alongside quantum cryptanalysis targeting classical public-key infrastructure;
- **Unstructured Search:** Grover's algorithm, which achieves a provable quadratic speedup over any classical search procedure;
- **Quantum Simulation:** Simulating physical quantum many-body systems characterized by exponentially scaling Hilbert spaces, a task fundamentally intractable for classical architectures;
- **Machine Learning:** High-dimensional quantum linear algebra, variational quantum algorithms, and kernel methods operating in high-dimensional feature spaces;
- **Computational Biology:** Biomolecular structure prediction, protein folding dynamics, and macromolecular docking simulations;
- **Generative Chemistry:** Electronic structure calculations, molecular ground-state estimation, and computational catalyst discovery.

Classical computing will naturally not be displaced. Rather, classical and quantum paradigms exhibit distinct, highly complementary operational profiles. While conventional processors continue to advance through architectural specialization and nanoscale semiconductor fabrication, physical quantum processors remain extraordinarily challenging to engineer, isolate, and scale.

Presently, the predominant operational model for utilizing physical quantum hardware is resource sharing via cloud-based computing infrastructures. However, these engineering hurdles do not diminish the practical and theoretical value of quantum algorithm design and software engineering. On the contrary, quantum computational resources scale fundamentally differently across specific complexity classes, enabling quantum processors to achieve exponential or super-polynomial advantages over the best-known classical algorithms for structured problems. This advantage stems directly from the fact that quantum computational primitives—principally coherent superposition, quantum interference, and multi-particle entanglement—are inaccessible to classical physics. Conversely, quantum computing is not an omnipotent remedy: many computational problems remain intractable regardless of whether classical or quantum resources are deployed.

Historically, research in this domain was confined almost exclusively to theoretical physicists and mathematicians equipped to navigate the formalisms of quantum mechanics. This dynamic has shifted dramatically in recent years: industry and academia have engineered high-level software development frameworks, domain-specific programming languages, and simulation runtimes. These tools abstract away low-level Hamiltonian engineering, enabling computer scientists and software engineers to design, verify, and simulate quantum algorithms programmatically.

1.1 Definition of a Quantum Computer

To delineate the boundaries of quantum computation, we first define a computer in foundational terms as a physical apparatus that ingests structured input data, executes systematic state transformations governed by physical laws, and yields deterministic or probabilistic outputs. Conventional digital computers—ranging from microcontrollers to supercomputing clusters—are grounded entirely within the theoretical framework

of classical physics, operating in accordance with Newtonian mechanics and Maxwellian electrodynamics, realized physically through semiconductor gating and macroscopic electron transport.

Within this framework, a **quantum computer** is defined as a computational device that processes information by exploiting physical phenomena governed strictly and exclusively by the laws of quantum mechanics.

Accordingly, the dichotomy between classical and quantum computing mirrors the fundamental distinction between classical and quantum physics. The operational divergence manifests in the physical mechanisms governing information representation and state evolution: where classical registers represent discrete strings of deterministic bits $x \in \{0,1\}^n$, a quantum register represents statevectors in a 2^n-dimensional complex Hilbert space $\mathcal{H}_{2^n} = \left(\mathbb{C}^2\right)^{\otimes n}$. In physical implementations, quantum processors are orchestrated via classical control systems that generate microwave pulses or laser pulses and convert projective measurement outcomes into classical bitstreams readable by standard CPUs.

Fundamentally, a quantum computer differs from its classical counterpart in its capacity to circumvent classical computational time-complexity bottlenecks for specific problem classes. When addressing computationally intensive problems where heuristics are inadmissible and classical silicon architectures approach physical scaling limits, adopting a quantum mechanical computational model offers an exponential speedup or polynomial acceleration that cannot be replicated by simply scaling classical hardware.

1.2 Hardware Limits

Currently, software developed for quantum computing can be executed either on classical software-based simulators or on physical quantum processors accessed remotely via cloud platforms. This operational bifurcation is dictated by the extreme physical engineering requirements of quantum hardware, which preclude deployment in conventional desktop or data-center environments.

Specifically, physical qubits are acutely susceptible to environmental perturbations, thermal fluctuations, electromagnetic radiation, and cosmic rays, all of which induce **quantum decoherence** and gate infidelity. Mitigating these errors requires isolating the physical processor (e.g., superconducting transmon circuits or trapped ions) inside dilution refrigerators maintained at cryogenic temperatures approaching absolute zero ($T \approx 10 - 20 \ \ \mathrm{mK}$). Such thermodynamics demand complex helium-dilution infrastructures that cannot be maintained outside specialized laboratory facilities.

Consequently, physical quantum processors remain primarily accessible through managed cloud computing infrastructures. Over recent years, major commercial and academic platforms have emerged:

- **IBM Quantum:** A pioneer in providing cloud access to superconducting transmon processors via the Qiskit software development kit;
- **Google Quantum AI:** Operating high-fidelity superconducting QPUs (Sycamore) alongside advanced simulation tooling;
- **Xanadu Quantum Cloud:** Photonic quantum computing offering continuous-variable and qubit-based architectures via the Strawberry Fields and PennyLane frameworks;
- **Amazon Braket:** A managed quantum computing service on AWS providing heterogeneous access to varied quantum architectures, including trapped-ion (IonQ), superconducting (Rigetti, Oxford Quantum Circuits), and neutral-atom systems (QuEra).

These platforms represent the primary commercial avenues for quantum execution, among a growing ecosystem of independent hardware providers.

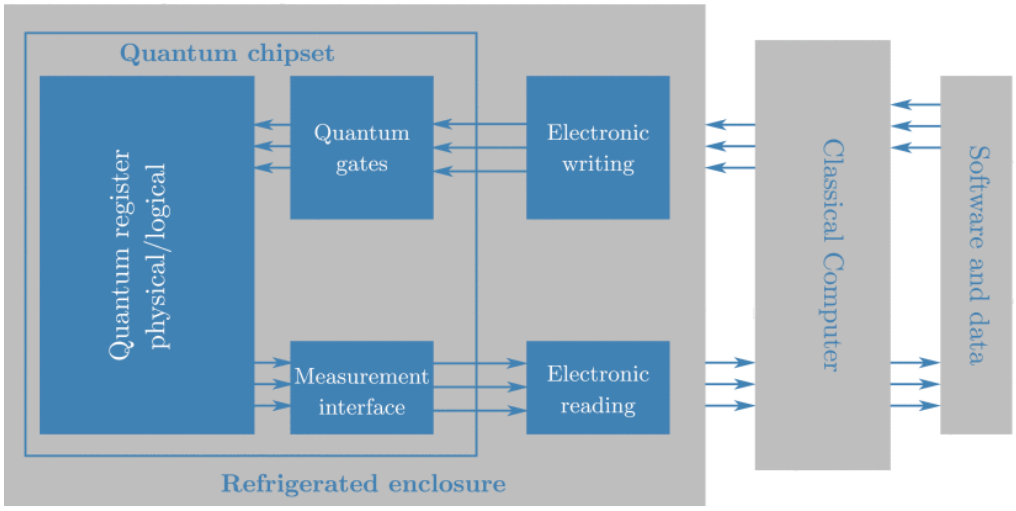


Figure 1: Hierarchical architecture of a quantum computer — adapted from an article in the IEEE Xplore Journal (https://ieeexplore.ieee.org/document/9274431).

Software simulators and physical quantum hardware share common abstraction layers and interfaces, enabling developers to formulate and validate algorithms prior to deployment on physical QPUs. Simulators explicitly represent the full quantum statevector in classical memory, providing complete observability at the expense of exponential memory growth ($\mathcal{O}(2^n)$ complex amplitudes for n qubits). A prominent example is `qsim`, developed by Google Quantum AI, which provides highly optimized, multi-threaded vector-register simulation of quantum circuits up to approximately 20–30 qubits in conjunction with the Cirq framework.

1.3 Technological Applications and Computational Complexity

Quantum computing establishes a fundamentally new taxonomy of computational complexity. Theoretical research has identified canonical algorithmic paradigms that exploit superposition, quantum interference, and entanglement to achieve decisive separations over classical computing:

- **Grover's Algorithm:** Solves the problem of unstructured search over a database or solution space of N items in $\mathcal{O}\left(\sqrt{N}\right)$ query steps, providing a provable quadratic speedup over the classical deterministic and randomized lower bound of $\Omega(N)$.
- **Shor's Algorithm:** Efficiently computes prime factorizations and discrete logarithms in polynomial time $\tilde{\mathcal{O}}((\log N)^3)$. In contrast, the most efficient known classical algorithm, the General Number Field Sieve (GNFS), scales in sub-exponential time:

  $$\exp\left((c + o(1))(\ln N)^{\frac{1}{3}}(\ln \ln N)^{\frac{2}{3}}\right)$$

  This polynomial-time solution renders vulnerable widespread public-key cryptosystems (such as RSA, Diffie-Hellman, and elliptic-curve cryptography), driving international migration toward Post-Quantum Cryptography (PQC).
- **Deutsch-Jozsa Algorithm:** Evaluates whether a black-box Boolean function is constant or balanced in deterministic constant time $\mathcal{O}(1)$, in contrast to classical deterministic algorithms which require $2^{n-1} + 1$ queries in the worst case ($\mathcal{O}(2^n)$).
- **Simon's Algorithm:** Determines the hidden bit-string period s of a two-to-one function $f(x) = f(x \oplus s)$ in polynomial time $\mathcal{O}(n)$, exhibiting an exponential query separation over classical randomized algorithms requiring $\Omega\left(2^{\frac{n}{2}}\right)$ evaluations. Simon's algorithm provided the direct conceptual blueprint for Shor's quantum Fourier transform.

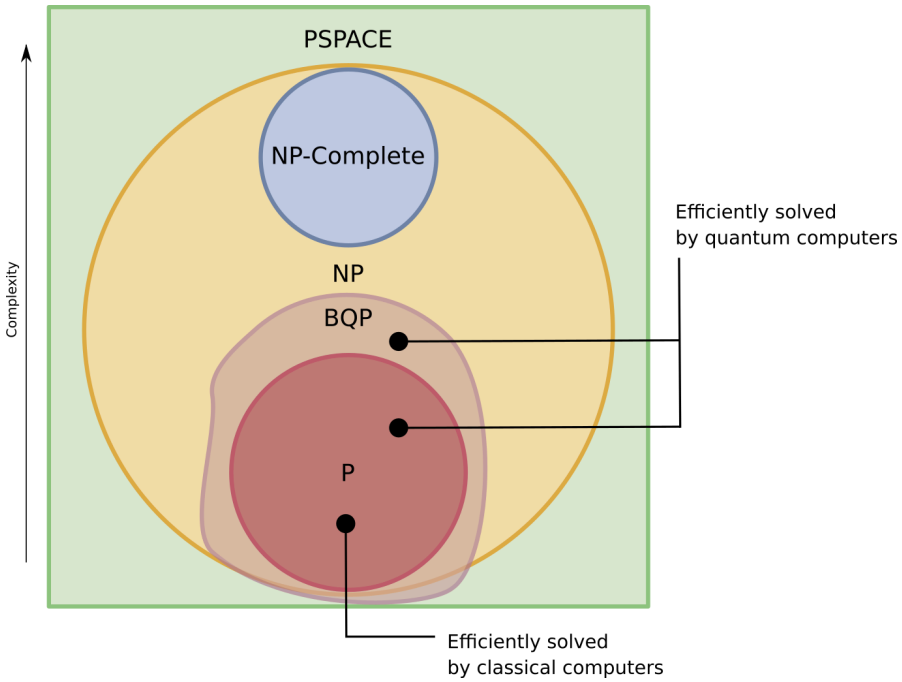


Figure 2: Relationships among fundamental computational complexity classes. BQP denotes Bounded-Error Quantum Polynomial Time, capturing problems efficiently solvable on a quantum computer.

The relationship between classical and quantum computational power is formalized through structural complexity classes:

- P denotes decision problems solvable in deterministic polynomial time on a classical Turing machine.
- NP consists of decision problems whose solutions are verifiable in classical polynomial time.
- BPP (Bounded-Error Probabilistic Polynomial Time) denotes problems solvable efficiently by randomized classical algorithms with error probability bounded below $\frac{1}{3}$.
- BQP (Bounded-Error Quantum Polynomial Time) represents the class of decision problems solvable by a quantum computer in polynomial time with error probability at most $\frac{1}{3}$.
- EQP (Exact Quantum Polynomial Time) denotes problems solvable by a quantum computer in polynomial time with **zero** error probability.

As illustrated in Figure 2, it is known that $P \subseteq BPP \subseteq BQP$. The exact boundaries between BQP and NP remain one of the central open questions in theoretical computer science. While Shor's algorithm proves that integer factorization resides in BQP, factorization is in $NP \cap \text{co-}NP$ and is not believed to be NP-complete. The prevailing consensus conjectures that $NP \not\subseteq BQP$ and $BQP \not\subseteq NP$, meaning quantum computers can efficiently solve certain problems outside P that are intractable for classical computers, yet cannot efficiently solve general NP-complete problems.

Beyond these foundational paradigms, quantum algorithms find impactful application across specialized scientific domains:

- **Gauss Sum Estimation:** Estimating exponential Gauss sums with polynomial precision in polynomial time via the quantum Fourier transform, contrasting with the exponential time demanded classically;
- **Boolean Formula Evaluation:** Accelerating the evaluation of highly complex Boolean formulas and tree structures through continuous-time and discrete-time quantum walks;
- **Quantum Many-Body Simulation:** Simulating Hamiltonian dynamics in condensed-matter physics and quantum chemistry (via Trotter-Suzuki decompositions and Variational Quantum Eigensolvers), unlocking precise modeling of material properties and chemical kinetics;

- **True Quantum Random Number Generation (QRNG):** Classical pseudo-random number generators (PRNGs) are fundamentally deterministic. By exploiting the projective measurement of qubits in balanced superposition states, quantum devices generate true nondeterministic entropy governed by Born's rule ($P(i) = |\langle i|\psi\rangle|^2$).

Conversely, identifying quantum speedups in classical machine learning remains a nuanced frontier. Classical machine learning often depends on high-bandwidth random access to massive tabular datasets. Transforming large classical datasets into quantum state amplitudes requires quantum random access memory (QRAM), whose physical overhead can offset algorithmic advantages. Consequently, computational tasks characterized by moderate input/output volumes but intensive algebraic transformations represent the most promising candidates for quantum acceleration.

2 Environment and Tooling

A diverse array of development environments and software development kits (SDKs) has emerged to facilitate quantum software engineering. Prominent platforms include Microsoft Azure with its Quantum Development Kit (QDK) and IBM's Qiskit framework. Additional SDKs utilized to construct, optimize, and simulate quantum circuits or dispatch jobs to physical quantum processors include:

- **Ocean** (D-Wave Systems) — specialized for quantum annealing;
- **ProjectQ** (ETH Zürich) — an open-source modular compiler framework;
- **Forest** / pyQuil (Rigetti Computing) — quantum instruction language and simulator suite;
- **t|ket>** (Quantinuum / Cambridge Quantum) — retargetable optimizing quantum compiler;
- **Strawberry Fields** (Xanadu) — photonic continuous-variable quantum computing;
- **PennyLane** (Xanadu) — hybrid quantum-classical machine learning and automatic differentiation.

The majority of these projects are open-source and built around Python ecosystems. For the investigations presented in this thesis, we selected the tools provided by Microsoft due to their comprehensive documentation and because the Microsoft QDK serves as the primary instructional foundation in our reference text, *Learn Quantum Computing with Python and Q#* by Sarah C. Kaiser and Christopher Granade.

The experimental work for this thesis was conducted in a Linux operating system environment; the installation workflow is completely analogous under Windows using PowerShell and `conda` or `pip`.

The Q# development environment can be configured within the Visual Studio Code editor via Microsoft's Quantum Development Kit extension. This extension integrates seamlessly with the compiler and local execution runtimes, providing syntax highlighting, IntelliSense, and integrated debugging.

In parallel, Q# code can be developed, executed, and documented interactively within Jupyter Notebooks powered by a Python backend. This workflow requires the appropriate .NET execution runtime. For our implementations, we utilized the long-term support (LTS) release of .NET Core 6.0, available directly through Microsoft's distribution channels or via standard Linux distribution package managers.

Executing Q# code within Jupyter Notebooks also requires the ASP.NET Core runtime. Once the .NET runtime is installed, the IQ# kernel can be installed and registered globally:

```
$ dotnet tool install -g Microsoft.Quantum.IQSharp
$ dotnet iqsharp install
```

In certain Linux configurations, installing the kernel per-user is required:

```
$ dotnet iqsharp install --user
```

This procedure registers the IQ# execution kernel, enabling Jupyter Notebooks to parse and execute Q# code blocks natively.

To streamline the management of Python dependencies, compiler toolchains, and scientific computing packages, utilizing a pre-configured distribution such as Anaconda is advantageous. If necessary, the Anaconda binary directory should be appended to the system environment path:

```
$ export PATH=/opt/anaconda/bin:$PATH
```

Using the included Conda package manager, an isolated virtual environment containing all necessary components can be created and activated:

```
$ conda create -n qsharp-env -c microsoft qsharp notebook
$ conda activate qsharp-env
```

This environment can be activated at any time to provide all runtimes needed to execute our hybrid Q# and Python applications. The software versions used in this work correspond to Python 3.9+ and .NET 6.0 LTS.

Local execution of Q# programs can be orchestrated through Python using the `qsharp` interop package.

```
import qsharp
from QsharpNamespace import Operation_One, Operation_Two

var1 = 10
print("Simulation started...")
Operation_One.simulate(par1=var1)
Operation_Two.simulate(par2=var1, par3=5)
```

Listing 1: `host.py` — Orchestrating Q# operations from a Python driver script.

As shown in Listing 1, an external Python driver script `host.py` initiates a local quantum simulator session and calls operations (`Operation_One`, `Operation_Two`) defined in companion `.qs` source files. The `qsharp` package automatically inspects the local directory, compiles the Q# source files, and exposes them as callable Python objects. This exemplifies the hybrid quantum-classical computing paradigm: the classical host handles orchestration, data pre-processing, and iterative control flow, delegating quantum kernel execution to the simulator.

Alternatively, Q# operations can be defined inline directly inside Python scripts using the `qsharp.compile` API:

```
import qsharp

prepare_qubit = qsharp.compile("""
open Microsoft.Quantum.Diagnostics;

operation PrepareQubit() : Unit {
    use qubit = Qubit();
    DumpMachine();
}
""")

if __name__ == "__main__":
    prepare_qubit.simulate()
```

Listing 2: `qsharp-interop.py` — Inline compilation and simulation of Q# code in Python.

In Listing 2, the diagnostic operation `DumpMachine()` from `Microsoft.Quantum.Diagnostics` inspects the statevector of the quantum simulator. Unlike physical quantum processors—where reading the internal statevector without wavefunction collapse is fundamentally forbidden by quantum mechanics—classical simulators maintain the full complex amplitude vector in memory, allowing `DumpMachine()` to output the complete probability amplitudes and phases across all basis states for debugging purposes.

With these foundational tools configured, one can execute and study quantum algorithms and explore the codebase provided by the authors of *Learn Quantum Computing with Python and Q#*.

3 The Q# Programming Language

To establish a rigorous foundation for the implementations presented in subsequent chapters, this chapter introduces the core syntax, operational paradigms, and type system of the Q# programming language.

Q# is an open-source, high-level, domain-specific programming language developed by Microsoft as the centerpiece of the Quantum Development Kit (QDK). Q# blends paradigms from modern imperative and functional languages (such as C#, F#, and Python), supporting classical control structures including loops, conditional branching (`if`/`then`/`else`), and structured data collections.

Crucially, Q# introduces domain-specific constructs tailored specifically for quantum computation, such as the `repeat ... until ... fixup` loop (used in repeat-until-success protocols) and built-in quantum phase estimation primitives. The language is strictly hardware-agnostic: a Q# program describes a quantum computation independently of whether it is dispatched to a local statevector simulator, a Clifford stabilizer emulator, a resource estimator, or a physical quantum processing unit (QPU).

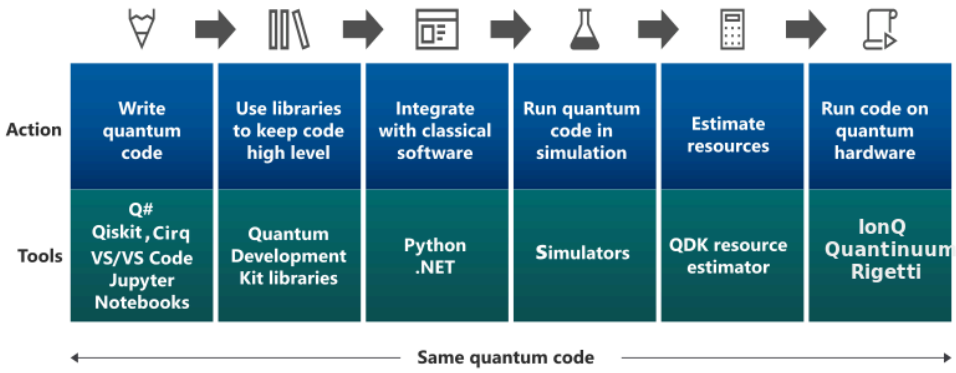


Figure 3: Workflow diagram illustrating the compilation and execution pipeline within the Microsoft QDK ecosystem, translating high-level Q# code through intermediate representations to quantum execution targets.

A minimal introductory Q# program is structured as follows:

```
namespace HelloQuantum {
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Intrinsic;

    @EntryPoint()
    operation SayHelloQ() : Unit {
        Message("Hello quantum world!");
    }
}
```

This snippet prints the message `"Hello quantum world!"` to the standard output. The `@EntryPoint()` attribute designates the routine as the execution starting point for the compiler.

In addition to standard classical data types (`Int`, `Double`, `Bool`, `String`), Q# provides types unique to quantum information:

- `Qubit`: An opaque reference handle representing an addressable two-level quantum subsystem in a Hilbert space $\mathcal{H}_2$;

- `Result`: Represents the classical outcome of a projective quantum measurement, restricted to two discrete values: `Zero` (eigenvalue $+1$) and `One` (eigenvalue -1);
- `Pauli`: Enumerates the single-qubit Pauli observables (`PauliI`, `PauliX`, `PauliY`, `PauliZ`).

While developers can declare user-defined types (records/tuples), Q# intentionally omits traditional object-oriented classes and inheritance in favor of modular operations and functions.

Qubits are allocated using the `use` statement, ensuring proper lifecycle management:

```
use q = Qubit();
// Allocation of an array of n qubits:
use register = Qubit[n];
```

Every qubit allocated via `use` is guaranteed by the Q# runtime to be initialized in the computational basis ground state $|0\rangle$. Crucially, the runtime enforces a strict safety invariant: prior to exiting the scope of the `use` block, every allocated qubit must be returned to the $|0\rangle$ state (via measurement and conditional correction, or through uncomputation). Violating this invariant causes a runtime exception to prevent unintended entanglement leakage.

In Q#, computational routines are divided into two distinct categories:

1. **Operations:** Callables that may execute quantum operations, modify quantum statevectors, allocate qubits, or perform nondeterministic projective measurements.
2. **Functions:** Pure, deterministic classical routines that have no quantum side-effects and cannot manipulate qubits directly.

Furthermore, Q# operations can declare support for **functors**—meta-operations that synthesize modified versions of an operation:

- `Adjoint`: Automatically synthesizes the inverse unitary operator $U^\dagger = (U^T)^*$. Because all non-measurement quantum evolutions are unitary and reversible, the `Adjoint` functor is essential for automatic uncomputation in quantum algorithms.
- `Controlled`: Synthesizes a controlled variant of the unitary operator, $C(U)$, conditioned on an array of control qubits.
- Operations supporting both functors declare the signature `is Adj + Ctl`.

```
operation SayHelloQ() : Unit {
    Message("Hello quantum world!");
}
```

A cornerstone of any quantum algorithm is the projective measurement of qubits in a designated measurement basis:

```
operation MeasureOneQubit() : Result {
    // Allocate a single qubit, initialized to |0⟩ by convention
    use q = Qubit();

    // Apply a Hadamard gate to prepare a balanced superposition state
    // |+⟩ = 1/√2 (|0⟩ + |1⟩)
    H(q);

    // Measure the qubit in the standard computational (Pauli-Z) basis
    let result = M(q);

    // Qubits must always be reset to |0⟩ prior to deallocation
    if result == One {
        X(q);
    }

    return result;
}
```

In the listing above, a qubit is allocated in the $|0\rangle$ state, transformed into the superposition state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ via the Hadamard gate `H`, and measured using the `M` operation. The `M` callable measures a single qubit in the Pauli-Z basis ($\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$), returning `Zero` or `One` with equal probability (50%). Calling `M(q)` is semantically equivalent to `Measure([PauliZ], [q])`.

Because the Q# runtime strictly requires qubits to be returned to the $|0\rangle$ state before release, resetting operations are ubiquitous. The standard library provides convenient composite operations such as `MResetZ` and `MResetX`, which measure and reset a qubit in a single atomic call:

```
operation MResetX (target : Qubit) : Result
```

Here, `MResetX` measures the target qubit in the Pauli-X eigenbasis $\{|+\rangle, |-\rangle\}$ and resets it to $|0\rangle$. Mathematically, measuring in the X-basis corresponds to applying a Hadamard transformation followed by a computational Z-basis measurement ($M_X = HM_ZH$).

Once measured, quantum information collapses into classical bits (`Result`), allowing subsequent control flow in the calling classical environment to proceed using ordinary deterministic logic.

4 Quantum Oracles

To implement and evaluate algorithms such as Deutsch-Jozsa, we must construct quantum representations —termed *quantum oracles*—of the mathematical functions supplied as input.

Definition (Quantum Oracle): A quantum oracle U_f associated with a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is a unitary operator that evaluates f conditionally upon basis state labels. In the canonical phase-kickback and bit-flip embeddings, applying the oracle twice yields the identity operator:

$$U_f^2 = I$$

Since U_f is unitary ($U_f^\dagger U_f = I$), $U_f^2 = I$ implies $U_f^\dagger = U_f$, demonstrating that Boolean quantum oracles are self-adjoint (Hermitian) unitary transformations.

Because closed quantum evolution is governed by Schrödinger's equation, all quantum gate operations (excluding measurement) must be strictly unitary and therefore reversible. Classical Boolean functions, however, are frequently irreversible: for example, a constant function $f(x) = 0$ discards all information regarding its input x, precluding any reconstruction of x from the output.

To resolve this fundamental incompatibility, irreversible classical functions $f : \{0,1\}^n \to \{0,1\}$ must be embedded into reversible bijective functions $h : \{0,1\}^n \times \{0,1\} \to \{0,1\}^n \times \{0,1\}$:

$$h(x, y) = (x, y \oplus f(x))$$

where $\oplus$ denotes addition modulo 2 (the bitwise XOR operation). This formulation preserves the input register x intact while writing the function evaluation into an auxiliary target register y.

This principle translates directly into the definition of a quantum oracle U_f:

$$U_f \ |x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

Here, $|x\rangle$ is the n-qubit query register and $|y\rangle$ is the single-qubit target register. Because $y \oplus f(x) \oplus f(x) = y$, applying U_f a second time restores the initial state $|x\rangle|y\rangle$, confirming unitarity: $U_f^\dagger U_f = I$.

To synthesize these oracles on quantum hardware, we employ primitive quantum gates:

- **Pauli-X gate:** The quantum analogue of the classical NOT gate:

$$X \ |0\rangle = |1\rangle, \quad X \ |1\rangle = |0\rangle$$

Represented in matrix form as:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- **Controlled-NOT (CNOT) gate:** Inverts the target qubit t if and only if the control qubit c is in state $|1\rangle$:

$$\text{CNOT} \ |00\rangle = |00\rangle, \quad \text{CNOT} \ |01\rangle = |01\rangle$$

$$\text{CNOT} \ |10\rangle = |11\rangle, \quad \text{CNOT} \ |11\rangle = |10\rangle$$

In matrix form across the two-qubit computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- **SWAP gate:** Exchanges the quantum states of two qubits:

$$\text{SWAP } |01\rangle = |10\rangle, \quad \text{SWAP } |10\rangle = |01\rangle$$

  Represented in matrix form as:

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The Q# implementations of single-qubit and multi-qubit oracles evaluated in our experiments are detailed in Listing 3:

```
namespace QuantumOracles {
    open Microsoft.Quantum.Intrinsic;

    operation ApplyZeroOracle(control : Qubit, target : Qubit) : Unit {
        // f1(x) = 0: Target remains unchanged (Identity operation)
    }

    operation ApplyOneOracle(control : Qubit, target : Qubit) : Unit {
        // f2(x) = 1: Target is flipped unconditionally
        X(target);
    }

    operation ApplyZeroOracleN(control : Qubit[], target : Qubit) : Unit {
        // n-qubit constant zero
    }

    operation ApplyOneOracleN(control : Qubit[], target : Qubit) : Unit {
        // n-qubit constant one
        X(target);
    }

    operation ApplyIdOracle(control : Qubit, target : Qubit) : Unit {
        // f3(x) = x: CNOT computes target = target ⊕ control
        CNOT(control, target);
    }

    operation ApplyXOROracleN(control : Qubit[], target : Qubit) : Unit {
        // f5(x) = ⊕_i x_i: Apply CNOT across all control qubits
        for qubit in control {
            CNOT(qubit, target);
        }
    }

    operation ApplyNotOracle(control : Qubit, target : Qubit) : Unit {
        // f4(x) = ¬x: Invert control, apply CNOT, restore control
        X(control);
        CNOT(control, target);
        X(control);
    }
}
```

Listing 3: `oracles.qs` — Implementation of constant and balanced quantum oracles in Q#.

These oracles represent Boolean functions of the general signature:

$$f : \{0,1\}^n \to \{0,1\}$$

Specifically, we defined oracles for five canonical test functions:

- $f_1(x) = 0$ (Constant zero);
- $f_2(x) = 1$ (Constant one);
- $f_3(x) = x$ (Balanced identity function);
- $f_4(x) = \neg x = 1 - x$ (Balanced bitwise negation);
- $f_5(x) = \bigoplus_{i=0}^{n-1} x_i$ (Balanced parity / multi-qubit XOR sum).

The first two functions (f_1, f_2) are strictly constant; the remaining three (f_3, f_4, f_5) are balanced.

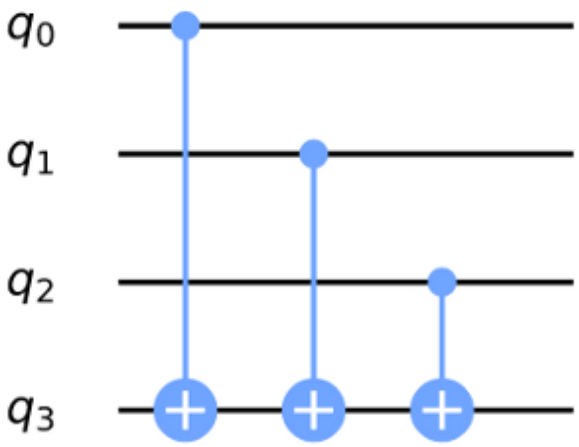


Figure 4: Example of a balanced 4-qubit quantum parity oracle implemented with cascaded CNOT gates, computing $q_3 = q_3 \oplus q_0 \oplus q_1 \oplus q_2$.

For an arbitrary n-qubit input $x = (x_0, x_1, ..., x_{n-1})$, the quantum oracle action generalizes to:

$$U_f \ |x_0 x_1 ... x_{n-1} y\rangle = |x_0 x_1 ... x_{n-1}\rangle \otimes |f(x_0, x_1, ..., x_{n-1}) \oplus y\rangle$$

In computational complexity theory, the concept of an *oracle machine* originated with Alan Turing (1939). An oracle machine is formalized as a Turing machine equipped with an auxiliary "black-box" tape capable of deciding membership in a designated language B in a single computational step. In quantum query complexity, oracles serve as black-box subroutines, allowing researchers to evaluate the query complexity —the number of calls to U_f required to evaluate a global property of f—independently of specific gate implementations.

5 The Deutsch-Jozsa Algorithm

The Deutsch-Jozsa algorithm holds seminal historical importance: it was the first quantum algorithm demonstrated to achieve an exponential separation in query complexity over deterministic classical computation, providing definitive proof-of-concept for the computational power of quantum information processing.

The algorithm addresses a promise problem concerning a black-box Boolean function:

$$f : \{0,1\}^n \to \{0,1\}$$

The algorithm operates under the explicit promise that f belongs to one of two mutually exclusive categories:

- **Constant:** $f(x)$ returns the exact same value for all inputs ($f(x) = 0$ for all x, or $f(x) = 1$ for all x);
- **Balanced:** $f(x)$ evaluates to 0 for exactly half of all possible inputs (2^{n-1} states), and to 1 for the remaining half.

The objective is to determine with complete certainty whether f is constant or balanced using the minimum number of oracle evaluations.

From a structural complexity perspective, this algorithm establishes that the complexity class **EQP** (Exact Quantum Polynomial Time) is strictly separated from classical deterministic polynomial time **P** under relativized oracle models ($EQP^A \not\subseteq P^A$).

5.1 The Classical Solution

In classical deterministic computation, deciding whether f is constant or balanced requires evaluating the function $f(x)$ on distinct inputs sequentially.

In the best-case scenario, evaluating two distinct inputs might immediately yield opposing outputs:

$$f(0,0,...,0) = 0, \quad f(1,0,...,0) = 1$$

Because f is promised to be either constant or balanced, observing two distinct outputs guarantees that f cannot be constant. Hence, the function is proven to be balanced in just $k = 2$ queries.

In the worst-case scenario, however, every queried input returns the identical value (e.g., 0). To establish with deterministic certainty that f is constant rather than balanced, a classical algorithm must inspect strictly more than half of the entire input domain. Because the domain contains 2^n distinct inputs, the deterministic query complexity is:

$$k_{\text{worst}} = \frac{2^n}{2} + 1 = 2^{n-1} + 1$$

Thus, classical deterministic algorithms scale as $\mathcal{O}(2^n)$, rendering exhaustive verification intractable as n grows.

If randomized bounded-error algorithms (BPP) are permitted, a probabilistic determination can be reached in k independent random queries. A randomized algorithm that evaluates k distinct random inputs, observes k identical evaluations, and guesses that f is constant has an error probability bounded by:

$$\varepsilon \leq \frac{1}{2^{k-1}}$$

While this error probability decays exponentially with k, achieving **deterministic zero-error certainty** (EQP) strictly requires $2^{n-1}+1$ evaluations classically.

5.2 The Quantum Solution

Originally conceived by David Deutsch and Richard Jozsa in 1992 and streamlined into the modern circuit model by Cleve, Ekert, Macchiavello, and Mosca in 1998, the quantum solution resolves the decision problem with **a single query** ($\mathcal{O}(1)$ query complexity) to the quantum oracle U_f, with zero probability of error.

The algorithmic procedure proceeds as follows:

1. Initialize two quantum registers: an n-qubit query register initialized to $|0...0\rangle = |0\rangle^{\otimes n}$, and a 1-qubit auxiliary target register initialized to $|1\rangle$.
2. Apply the Hadamard transform $H^{\otimes n}$ to the query register and H to the auxiliary register.
3. Apply the quantum oracle U_f, inducing a phase kickback onto the query register.
4. Discard the auxiliary register and apply $H^{\otimes n}$ again to the n-qubit query register.
5. Perform a computational basis measurement on the query register: measuring $|0...0\rangle$ proves deterministically that f is constant; measuring any state other than $|0...0\rangle$ proves deterministically that f is balanced.

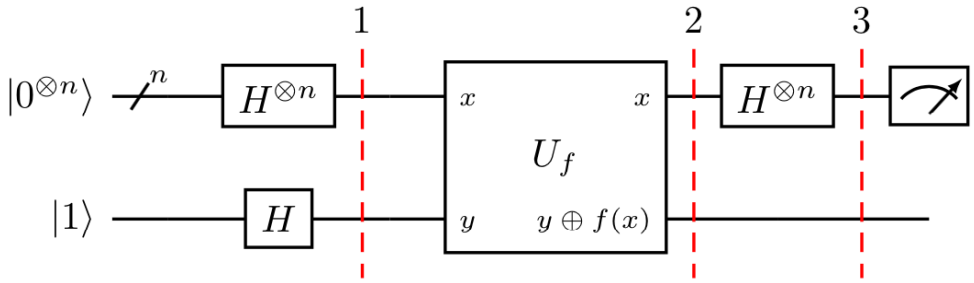


Figure 5: Quantum circuit diagram of the n-qubit Deutsch-Jozsa algorithm.

The foundational engine of the algorithm is the Hadamard transform H_m. The single-qubit Hadamard gate is represented by:

$$H_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

For an m-qubit register, the Hadamard transform $H_m = H^{\otimes m}$ can be expressed recursively starting from $H_0 = [1]$:

$$H_m = \frac{1}{\sqrt{2}}\begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix}$$

For instance, the 2-qubit transform is:

$$H_2 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Applying H to $|0\rangle$ and $|1\rangle$ creates balanced superposition states:

$$|+\rangle = H\ |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad |-\rangle = H\ |1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

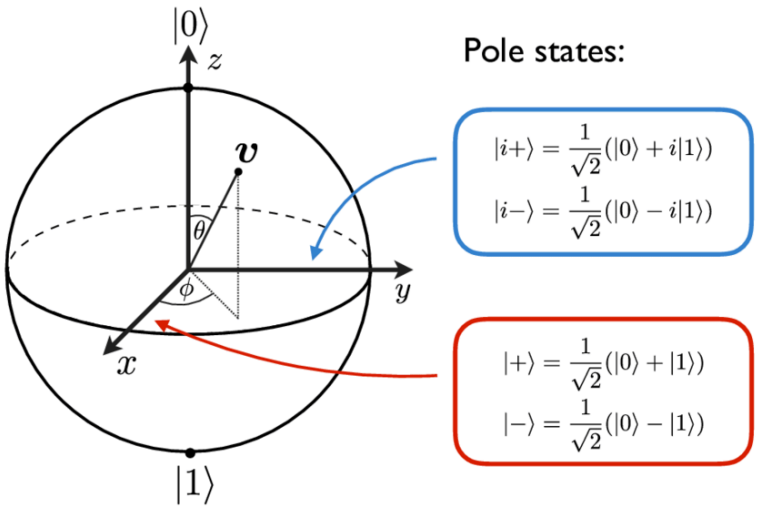


Figure 6: Bloch sphere representation of a single qubit. The computational basis states $|0\rangle$ and $|1\rangle$ align with the positive and negative z-axis poles, while superposition states $|+\rangle$ and $|-\rangle$ lie along the x-axis. The Hadamard gate corresponds to a π rotation around the $\frac{x+z}{\sqrt{2}}$ diagonal axis, mapping the z-axis onto the x-axis.

5.2.1 Single-Qubit Derivation ($n = 1$)

We trace the mathematical statevector evolution for $n = 1$:

1. **Initial state:**

$$|\psi_0\rangle = |0\rangle|1\rangle$$

2. **Superposition via Hadamard transform:**

$$|\psi_1\rangle = (H\ |0\rangle)(H\ |1\rangle) = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$$

3. **Oracle evaluation and Phase Kickback:** Recall the oracle definition: $U_f\ |x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$. When the target qubit is in the state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$:

$$U_f\ |x\rangle\ |-\rangle = \frac{1}{\sqrt{2}}(|x\rangle\ |0 \oplus f(x)\rangle - |x\rangle\ |1 \oplus f(x)\rangle) = (-1)^{f(x)}\ |x\rangle\ |-\rangle$$

   This phenomenon is the fundamental **quantum phase kickback**: the function evaluation $f(x)$ is transferred directly into the phase factor $(-1)^{f(x)}$ of the query state. Evaluating this across the superposition $|\psi_1\rangle$:

$$|\psi_2\rangle = \frac{1}{2}[(-1)^{f(0)}\ |0\rangle(|0\rangle - |1\rangle) + (-1)^{f(1)}\ |1\rangle(|0\rangle - |1\rangle)]$$

   Factoring out the global phase $(-1)^{f(0)}$ and the auxiliary state $|-\rangle$:

$$|\psi_2\rangle = \frac{(-1)^{f(0)}}{\sqrt{2}}[|0\rangle + (-1)^{f(0)\oplus f(1)}\ |1\rangle] \otimes |-\rangle$$

4. **Second Hadamard on query qubit:** Discarding the unentangled auxiliary qubit $|-\rangle$ and global phase, we apply H to the query qubit:

$$H\left[\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{f(0)\oplus f(1)}\ |1\rangle)\right] = \frac{1}{2}[(1 + (-1)^{f(0)\oplus f(1)})\ |0\rangle + (1 - (-1)^{f(0)\oplus f(1)})\ |1\rangle]$$

5. **Projective Measurement:**
   - If f is **constant**, $f(0) = f(1) \implies f(0) \oplus f(1) = 0$. The amplitude of $|1\rangle$ vanishes identically:

$$|\psi_3\rangle = \frac{1}{2}(2\ |0\rangle + 0\ |1\rangle) = |0\rangle$$

     Measuring the query qubit in the computational basis yields `Zero` with probability $P(|0\rangle) = 1$.
   - If f is **balanced**, $f(0) \neq f(1) \implies f(0) \oplus f(1) = 1$. The amplitude of $|0\rangle$ vanishes identically:

$$|\psi_3\rangle = \frac{1}{2}(0\ |0\rangle + 2\ |1\rangle) = |1\rangle$$

     Measuring the query qubit in the computational basis yields `One` with probability $P(|1\rangle) = 1$.

Thus, a single evaluation of the oracle determines the global property of f with zero error probability.

```
operation DeutschJozsaSingleBit(
    oracle : ((Qubit, Qubit) ⇒ Unit)
) : Bool {
    use control = Qubit();
    use target = Qubit();

    H(control);
    X(target);
    H(target);

    oracle(control, target);

    H(target);
    X(target);

    // Returns true if balanced, false if constant
    return MResetX(control) == One;
}
```

Listing 4: Single-qubit Deutsch-Jozsa algorithm implementation in Q#.

In Listing 4, the algorithm utilizes `MResetX(control) == One` to evaluate the final state. Because measuring an observable in the Pauli-X eigenbasis $\{|+\rangle, |-\rangle\}$ projects $|+\rangle$ onto `Zero` and $|-\rangle$ onto `One`, calling `MResetX` is functionally equivalent to applying a trailing Hadamard gate followed by a standard Z-basis measurement ($M_X = HM_ZH$), while atomically resetting the qubit to $|0\rangle$.

5.2.2 Generalization to n-Qubit Inputs

For an n-qubit input register, the system begins in $|0\rangle^{\otimes n} \otimes |1\rangle$. Applying $H^{\otimes(n+1)}$ prepares:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Applying U_f induces the phase kickback factor $(-1)^{f(x)}$ across every basis state in superposition:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}\ |x\rangle \otimes |-\rangle$$

Applying the n-qubit Hadamard transform $H^{\otimes n}$ to the query register utilizes the identity:

$$H^{\otimes n}\ |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} (-1)^{x \cdot y}\ |y\rangle$$

where $x \cdot y = \oplus_{i=0}^{n-1}\ x_i y_i$ represents the bitwise inner product modulo 2.

Substituting this transformation into the query register:

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{y=0}^{2^n-1} \left[\sum_{x=0}^{2^n-1} (-1)^{f(x) \oplus (x \cdot y)} \right]\ |y\rangle$$

Focusing on the amplitude of the ground state $|0...0\rangle$ (where $x \cdot 0 = 0$ for all x):

$$C_{(|0...0\rangle)} = \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)}$$

The probability of measuring the all-zero state $|0...0\rangle$ is:

$$P(|0...0\rangle) = |C_{(|0...0\rangle)}|^2 = |\frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)}|^2$$

- If f is **constant**, $(-1)^{f(x)}$ is identical ($+1$ or -1) for all 2^n terms:

  $$P(|0...0\rangle) = |\frac{1}{2^n}(\pm 2^n)|^2 = 1$$

  Constructive interference concentrates the entire wavefunction amplitude onto $|0...0\rangle$. Complete destructive interference suppresses all non-zero states: $P(y \neq 0) = 0$.
- If f is **balanced**, $(-1)^{f(x)}$ evaluates to $+1$ for exactly 2^{n-1} inputs and -1 for the remaining 2^{n-1} inputs:

  $$C_{(|0...0\rangle)} = \frac{1}{2^n}(2^{n-1} - 2^{n-1}) = 0 \Longrightarrow P(|0...0\rangle) = 0$$

  Destructive interference completely cancels the amplitude of $|0...0\rangle$. Consequently, measuring the register is guaranteed to yield a non-zero state ($y \neq 0...0$).

```
operation DeutschJozsa(
    size : Int,
    oracle : ((Qubit[], Qubit) ⇒ Unit)
) : Bool {
    use control = Qubit[size];
    use target = Qubit();

    ApplyToEachA(H, control);
    X(target);
    H(target);

    oracle(control, target);

    H(target);
    X(target);

    let result = MResetX(control[0]) == One;
    ResetAll(control);
    return result;
}
```

Listing 5: Multi-qubit Deutsch-Jozsa algorithm implementation in Q#.

In Listing 5, `ApplyToEachA(H, control)` applies the Hadamard transform in parallel across all control qubits. The candidate's routine inspects `control[0]` via `MResetX` for linear parity oracles ($f_5(x) = \oplus_i \, x_i$), where the single-qubit projection suffices. In the general n-qubit Deutsch-Jozsa formulation, all n qubits are measured: the function is decided to be constant if and only if all measured qubits evaluate to `Zero` ($|0^n\rangle$).

6 Quantum Teleportation

We conclude our algorithmic investigations by examining one of the most remarkable phenomena in quantum information science: **quantum entanglement** and its application in **quantum teleportation**.

The term *entanglement* (originally *Verschränkung*) was introduced by Erwin Schrödinger in 1935 to designate the inseparable state correlations exhibited by composite quantum systems governed by global conservation laws. An entangled state of multiple particles cannot be factored into a tensor product of individual single-particle states ($|\Psi\rangle_{AB} \neq |\psi\rangle_A \otimes |\varphi\rangle_B$). Entangled states maintain non-local correlations regardless of the physical distance separating the particles: measuring an observable on one subsystem instantaneously determines the corresponding observable on the entangled partner.

Two foundational theorems govern the processing of quantum information:

1. **The No-Cloning Theorem** (Wootters & Zurek, 1982; Dieks, 1982): Proves that unitary quantum mechanics strictly forbids the creation of an identical copy of an arbitrary unknown quantum state. If a unitary operator U could clone arbitrary states $|\psi\rangle$ and $|\varphi\rangle$, unitarity would require:

$$\langle\psi|\varphi\rangle = \langle\psi|\ \langle e|\ U^\dagger U\ |\varphi\rangle\ |e\rangle = (\langle\psi|\ \langle\psi|)(|\varphi\rangle\ |\varphi\rangle) = \langle\psi|\varphi\rangle^2$$

   which implies $\langle\psi|\varphi\rangle \in \{0, 1\}$, proving that only orthogonal states can be cloned.
2. **The Non-Discrimination Theorem:** States that two non-orthogonal quantum states cannot be distinguished with absolute certainty in a single physical measurement.

Furthermore, elementary particles of the same species exhibit fundamental quantum indistinguishability: interchanging two identical particles produces an indistinguishable physical state up to a global phase. Rather than treating particles as localized classical objects with individual identities, it is physically more accurate to describe entangled particles as localized excitations of a shared quantum field.

Remarkably, despite the no-cloning theorem, an unknown quantum state $|\psi\rangle$ held by particle A_1 can be **teleported** to a distant particle B using an entangled pair (A_2, B) shared between two parties, traditionally designated Alice and Bob.

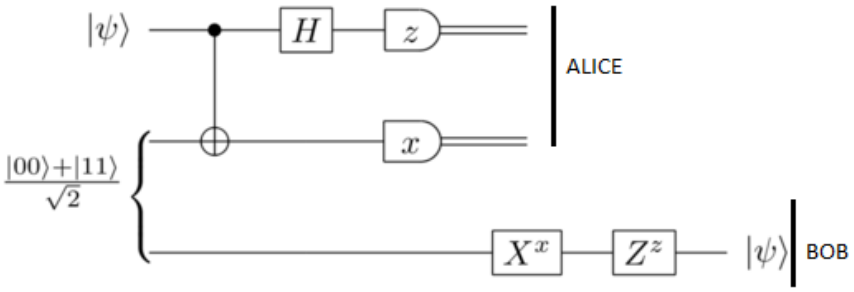


Figure 7: Quantum circuit implementing the standard quantum teleportation protocol. Alice performs a Bell-basis measurement on her two qubits; Bob applies feedforward unitary Pauli corrections conditioned on Alice's classical measurement results.

6.1 Mathematical Formulation

Let the arbitrary single-qubit state to be teleported be:

$$|\psi\rangle_{A_1} = \alpha\ |0\rangle_{A_1} + \beta\ |1\rangle_{A_1}$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. The coefficients α and β are unknown to both Alice and Bob.

Alice and Bob share an entangled Bell state (EPR pair):

$$|\Phi^+\rangle_{A_2B} = \frac{1}{\sqrt{2}}(|0\rangle_{A_2}\ |0\rangle_B + |1\rangle_{A_2}\ |1\rangle_B)$$

The initial three-qubit state of the entire system across registers A_1, A_2, B is:

$$|\Psi_0\rangle = |\psi\rangle_{A_1} \otimes |\Phi^+\rangle_{A_2B} = \frac{1}{\sqrt{2}}(\alpha\ |0\rangle_{A_1} + \beta\ |1\rangle_{A_1})(|0\rangle_{A_2}|0\rangle_B + |1\rangle_{A_2}|1\rangle_B)$$

Expanding the tensor product:

$$|\Psi_0\rangle = \frac{1}{\sqrt{2}}[\alpha\ |000\rangle + \alpha\ |011\rangle + \beta\ |100\rangle + \beta\ |111\rangle]_{A_1A_2B}$$

We express Alice's two qubits (A_1, A_2) in the orthonormal Bell basis:

$$|\Phi^\pm\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle), \quad |\Psi^\pm\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle)$$

Inverting these relations yields the computational basis states:

$$|00\rangle = \frac{1}{\sqrt{2}}(|\Phi^+\rangle + |\Phi^-\rangle), \quad |11\rangle = \frac{1}{\sqrt{2}}(|\Phi^+\rangle - |\Phi^-\rangle)$$

$$|01\rangle = \frac{1}{\sqrt{2}}(|\Psi^+\rangle + |\Psi^-\rangle), \quad |10\rangle = \frac{1}{\sqrt{2}}(|\Psi^+\rangle - |\Psi^-\rangle)$$

Substituting these expansions into $|\Psi_0\rangle$ and grouping terms according to Alice's Bell states:

$$\begin{aligned}|\Psi_0\rangle = \frac{1}{2}[&|\Phi^+\rangle_{A_1A_2} \otimes (\alpha\ |0\rangle_B + \beta\ |1\rangle_B) \\ &+ |\Phi^-\rangle_{A_1A_2} \otimes (\alpha\ |0\rangle_B - \beta\ |1\rangle_B) \\ &+ |\Psi^+\rangle_{A_1A_2} \otimes (\beta\ |0\rangle_B + \alpha\ |1\rangle_B) \\ &+ |\Psi^-\rangle_{A_1A_2} \otimes (-\beta\ |0\rangle_B + \alpha\ |1\rangle_B)]\end{aligned}$$

Alice performs a joint Bell-state measurement on her two qubits (A_1, A_2). In the quantum circuit (Figure 7), this is implemented by applying a CNOT gate from A_1 to A_2 followed by a Hadamard gate on A_1, transforming the Bell basis into the standard computational basis:

$$|\Phi^+\rangle \mapsto |00\rangle, \quad |\Phi^-\rangle \mapsto |10\rangle, \quad |\Psi^+\rangle \mapsto |01\rangle, \quad |\Psi^-\rangle \mapsto |11\rangle$$

Measuring Alice's two qubits yields two classical bits $(m_1, m_2) \in \{0, 1\}^2$, each occurring with equal probability (25%):

- If Alice measures $|\Phi^+\rangle$ (bits 00): Bob holds $\alpha\ |0\rangle + \beta\ |1\rangle = I\ |\psi\rangle$;
- If Alice measures $|\Phi^-\rangle$ (bits 10): Bob holds $\alpha\ |0\rangle - \beta\ |1\rangle = Z\ |\psi\rangle$;
- If Alice measures $|\Psi^+\rangle$ (bits 01): Bob holds $\beta\ |0\rangle + \alpha\ |1\rangle = X\ |\psi\rangle$;
- If Alice measures $|\Psi^-\rangle$ (bits 11): Bob holds $-\beta\ |0\rangle + \alpha\ |1\rangle = XZ\ |\psi\rangle$.

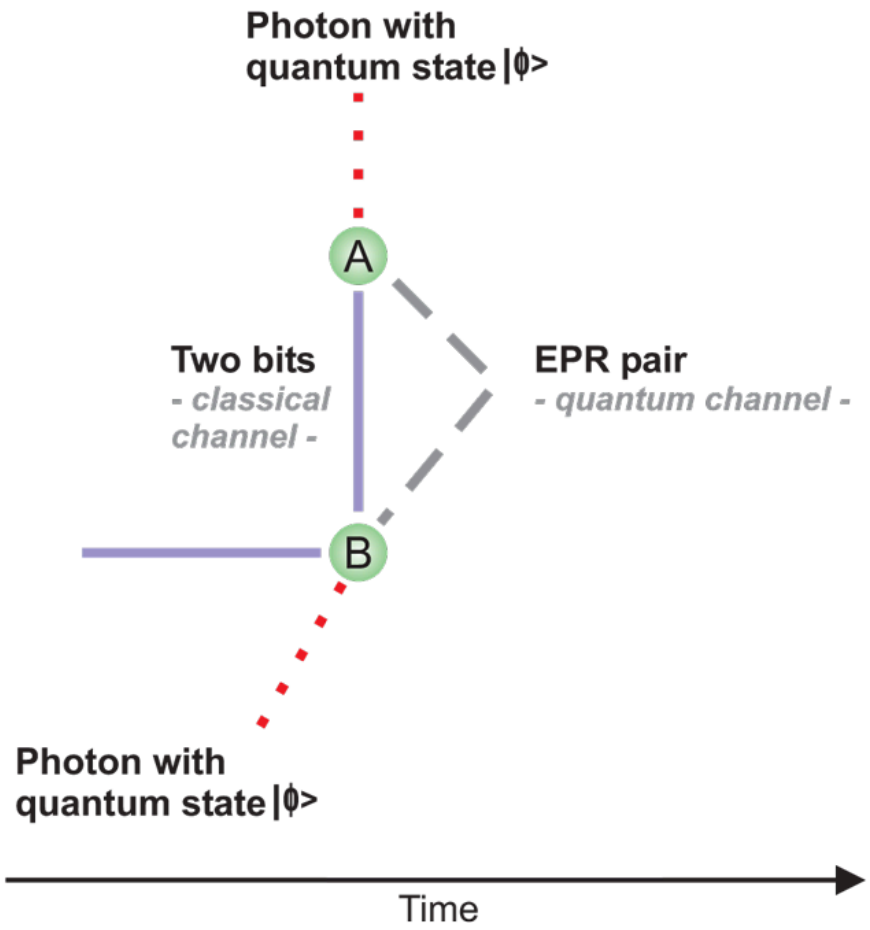


Figure 8: Quantum teleportation protocol combining a shared entangled quantum channel with a classical communication channel to transfer quantum states in full compliance with special relativity.

To recover the original state $|\psi\rangle$, Bob must know the outcome of Alice's measurement. Alice transmits her two classical measurement bits (m_1, m_2) to Bob across a conventional classical communication channel. Upon receipt, Bob applies the corresponding unitary feedforward correction:

$$U_{\text{Bob}} = X^{m_2} Z^{m_1} \in \{I, Z, X, XZ\}$$

which deterministically reconstructs the exact original state $|\psi\rangle$ at Bob's station.

This feedforward mechanism is fundamental: prior to receiving the classical measurement outcomes, Bob's local quantum state is described by the partial trace over Alice's qubits:

$$\rho_B = \text{Tr}_{A_1 A_2}(|\Psi_0\rangle\langle\Psi_0|) = \frac{1}{4}\sum_{k=1}^{4}|\psi_k\rangle\langle\psi_k| = \frac{1}{2}\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} = \frac{1}{2}I$$

Bob possesses a maximally mixed density matrix from which zero classical or quantum information can be extracted. Consequently, quantum teleportation cannot transmit information faster than the speed of light, strictly preserving relativistic causality. Furthermore, Alice's measurement irrevocably collapses and destroys the original state $|\psi\rangle_{A_1}$, ensuring full adherence to the no-cloning theorem.

Quantum teleportation has been validated across extensive experimental campaigns. A notable milestone was achieved across a 143 km free-space optical link between the Canary Islands of La Palma and Tenerife (Nature, 2012).

6.2 Implementation in Python and Q#

```
from interface import QuantumDevice, Qubit
from simulator import Simulator

def teleport(msg: Qubit, here: Qubit, there: Qubit) → None:
    # here and there are initialized to |0⟩
    # Prepare an entangled Bell pair between here (Alice) and there (Bob)
    here.h()
    here.cnot(there)

    # Alice interacts the message qubit with her entangled qubit
    msg.cnot(here)
    msg.h()

    # Measurement results are sent across a classical channel
    if msg.measure():
        there.z()
    if here.measure():
        there.x()

    msg.reset()
    here.reset()
```

Listing 6: Quantum teleportation protocol implementation in Python.

```
namespace Microsoft.Quantum.Samples.Teleportation {
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Measurement;

    operation Teleport(msg : Qubit, target : Qubit) : Unit {
        use register = Qubit();

        // Create Bell state between auxiliary register and target
        H(register);
        CNOT(register, target);

        // Perform Bell measurement on message and register
        CNOT(msg, register);
        H(msg);

        // Measure and apply conditional Pauli corrections
        if (IsResultOne(MResetZ(msg))) {
            Z(target);
        }
        if (IsResultOne(MResetZ(register))) {
            X(target);
        }
    }
}
```

Listing 7: Quantum teleportation routine in Q# (adapted from Microsoft QDK samples).

In the Q# implementation (Listing 7), Alice's auxiliary qubit is instantiated in the `register` variable. The library function `MResetZ` measures each qubit in the Pauli-Z basis and resets it to $|0\rangle$ in an atomic operation. The conditional statements `if (IsResultOne(...))` execute the classical feedforward corrections Z and X, perfectly reflecting the mathematical transformation $U_{\text{Bob}} = X^{m_2} Z^{m_1}$.

7 Conclusions

The field of quantum computation and quantum information processing is advancing at an unprecedented pace. While conceptually young—originating in the early 1980s with Paul Benioff's formulation of a quantum mechanical Hamiltonian model of a Turing machine, Richard Feynman's proposal for quantum simulation, and David Deutsch's universal quantum computing framework—it offers immense scientific and technological potential.

Recognizing its strategic significance, the European Union has designated quantum computing as a core technological pillar of the European Digital Decade, establishing concrete digitalization targets for member states to achieve by 2030. In conjunction, European nations signed the European Quantum Communication Infrastructure (EuroQCI) declaration to deploy a secure, continent-wide quantum communication backbone connecting quantum processors and secure communication nodes across Europe. These initiatives are essential for maintaining global technological competitiveness and bolstering cyber-resilience against emerging quantum threats.

A pivotal milestone in foundational quantum physics was recognized with the 2022 Nobel Prize in Physics, awarded to Alain Aspect, John Clauser, and Anton Zeilinger. Building upon John Bell's theoretical framework and the violation of Bell inequalities, their pioneering experiments with entangled photons conclusively ruled out local hidden-variable theories, experimentally validating the non-local nature of quantum mechanics and solidifying the conceptual bedrock upon which quantum information processing rests.

Simultaneously, classical computing is confronting inexorable physical limitations. For over five decades, processor performance scaled exponentially in accordance with Moore's Law, roughly doubling transistor density every two years, accompanied by Dennard scaling. Today, semiconductor fabrication processes operate at atomic dimensions where classical electrodynamics yields to quantum tunneling, leakage currents, and thermal dissipation bottlenecks. The scientific and industrial consensus indicates that the industry is entering a "post-Moore" era: physical scaling limits will be reached over the coming decades, shifting commercial innovation toward specialized domain-specific accelerators, energy-efficient mobile architectures, and sensor integration.

Consequently, developing fundamentally new computational paradigms has transitioned from a theoretical curiosity into an urgent engineering imperative. It is this technological reality that has accelerated global investment in quantum information systems—an endeavor that will continue to reshape computing as classical hardware approaches its ultimate physical boundaries.

8 Bibliography

1. Sarah C. Kaiser and Christopher Granade. *Learn Quantum Computing with Python and Q#*. Manning Publications, Shelter Island, NY, 2021. ISBN: 9781617296130. Available at: https://www.manning.com/books/learn-quantum-computing-with-python-and-q-sharp
2. J. S. Bell. “On the Einstein Podolsky Rosen Paradox”. *Physics Physique Fizika*, 1(3):195–200, 1964. DOI: 10.1103/PhysicsPhysiqueFizika.1.195. Available at: https://www.informationphilosopher.com/solutions/scientists/bell/Bell_On_EPR.pdf
3. S. Kumar. “Fundamental Limits to Moore’s Law”. arXiv preprint arXiv:1511.05956, 2015. Available at: https://arxiv.org/abs/1511.05956
4. M. Mitchell Waldrop. “The chips are down for Moore’s law”. *Nature*, 530(7589):144–147, 2016. DOI: 10.1038/530144a. Available at: https://www.nature.com/news/the-chips-are-down-for-moore-s-law-1.19338
5. Paul Benioff. “The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines”. *Journal of Statistical Physics*, 22(5):563–591, 1980. DOI: 10.1007/BF01011339. Available at: https://link.springer.com/article/10.1007/BF01011339
6. Sarah C. Kaiser and Christopher Granade. “Learn Quantum Computing with Python and Q# Code Repository”. GitHub, 2021. Available at: https://github.com/crazy4pi314/learn-qc-with-python-and-qsharp
7. IEEE Xplore Journal. “Quantum Computer System Architecture”. *IEEE Micro*, 40(6):1–10, 2020. DOI: 10.1109/MM.2020.3023456. Available at: https://ieeexplore.ieee.org/document/9274431
8. Wikipedia contributors. “Oracle machine”. *Wikipedia, The Free Encyclopedia*. Available at: https://en.wikipedia.org/wiki/Oracle_machine
9. Microsoft Corporation. “Azure Quantum: Install the Quantum Development Kit (QDK)”. Microsoft Learn, 2022. Available at: https://learn.microsoft.com/en-us/azure/quantum/install-overview-qdk
10. Microsoft Corporation. “Quantum Teleportation Sample”. Microsoft Quantum GitHub Samples, 2022. Available at: https://github.com/microsoft/quantum/tree/main/samples/getting-started/teleportation
11. Microsoft Corporation. “Quantum Computing Development Kit Resources”. Microsoft Azure, 2022. Available at: https://azure.microsoft.com/en-us/resources/development-kit/quantum-computing/
12. Qiskit Development Team. “Deutsch-Jozsa Algorithm”. *Qiskit Textbook*, 2021. Available at: https://qiskit.org/textbook/ch-algorithms/deutsch-jozsa.html
13. The Nobel Prize. “Press release: The Nobel Prize in Physics 2022”. Nobel Prize Outreach AB, Stockholm, 2022. Available at: https://www.nobelprize.org/prizes/physics/2022/press-release/