Accelerating Quantum Circuit Simulation with NVIDIA GPUs and tools

Maria Madalena Reforço Osório de Barros

2nd Cycle Integrated Project to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Aleksandar Ilić
Prof. Leonel Augusto Pires Seabra de Sousa

June 2025

Abstract

Quantum circuit simulation is essential for verifying quantum algorithms, especially in the context of Noisy Intermediate-Scale Quantum (NISQ) devices. As quantum hardware continues to develop, the demand for efficient and scalable simulation tools has grown, which are vital for the acceleration of quantum algorithm development and validating quantum software before it is deployed on quantum hardware. Classical simulation, particularly with GPU acceleration, has gained prominence as it allows for the simulation of larger quantum circuits with a greater number of qubits. This work benchmarks and, consequently, also evaluates the performance of two prominent quantum simulators, CUDA-Q and Qiskit Aer, across CPU and GPU backends.

Focusing on Quantum Fourier Transform (QFT) for the analysis of these simulators, the simulators were assessed based on computation time, scalability, and accuracy under different noise models. The findings highlight the trade-offs between memory usage, simulation time, and accuracy for both ideal and noisy quantum circuits. This PIC lays the groundwork for deeper investigations during my MSc thesis, where I plan to extend the benchmarking to additional simulators, algorithms, and further explore GPU-accelerated quantum simulation.

Keywords

Quantum Computing, Circuit Simulation, GPU Acceleration, CUDA-Q, Qiskit Aer

Resumo

A simulação de circuitos quânticos é essencial para a verificação de algoritmos quânticos, especialmente no contexto dos dispositivos Noisy Intermediate-Scale Quantum (NISQ). À medida que o hardware quântico continua a evoluir, a procura por ferramentas de simulação eficientes e escaláveis tem vindo a crescer, sendo fundamentais para acelerar o desenvolvimento de algoritmos quânticos e validar software quântico antes da sua execução em hardware real. A simulação clássica, particularmente com aceleração por GPU, tem ganho destaque por permitir a simulação de circuitos quânticos maiores, com um número mais elevado de qubits. Este trabalho avalia e compara o desempenho de dois simuladores quânticos de referência, o CUDA-Q e o Qiskit Aer, em plataformas com CPU e GPU.

Tendo como base de análise a Quantum Fourier Transform (QFT), os simuladores foram avaliados em termos de tempo de computação, escalabilidade e precisão, sob diferentes modelos de ruído. Os resultados evidenciam os compromissos entre utilização de memória, tempo de simulação e precisão, tanto em circuitos ideais como ruidosos. Este PIC constitui a base para uma exploração mais aprofundada na minha futura dissertação de mestrado, onde tenciono alargar o estudo a outros simuladores, algoritmos e técnicas de aceleração por GPU aplicadas à simulação de circuitos quânticos.

Palavras Chave

Computação Quântica, Simulação de Circuitos, Aceleração por GPU, CUDA-Q, Qiskit Aer

Contents

List of Acronyms

DFT Discrete Fourier Transform

GHZ Greenberger–Horne–Zeilinger

HSP Hidden Subgroup Problem

LSQb Least Significant Qubit

MPI Message Passing Interface

MPS Matrix Product State

NISQ Noisy Intermediate-Scale Quantum

QEC Quantum Error Correction

QFT Quantum Fourier Transform

RQC Random Quantum Circuit

SF Schrödinger–Feynman

SIMD Single Instruction Multiple Data

TN Tensor Network

CHAPTER 1
Introduction

Quantum computing promises transformative speed-ups in factoring, unstructured search, combinatorial optimisation, and some other areas, such as cryptanalysis. Yet commercial hardware remains in the Noisy Intermediate-Scale Quantum (NISQ) era, with prototype processors providing up to a few thousand physical qubits. Despite rapid growth, illustrated by IBM's *Condor* (1,121 qubits) and *Heron* (133 qubits per tile) modules two-qubit gate errors (10^{-3}) exceeds the (10^{-4}) threshold needed for surface code fault tolerance [1–4]. This noise and limited connectivity still constrain practical circuit depth, delaying fully error-corrected quantum advantage.

Until scalable, fault-tolerant hardware arrives, high-performance classical simulators remain indispensable. However, resource costs are high: simulating 30 30-qubit pure state requires approximately 16 GiB of memory in double precision, with additional overhead from Kraus-map noise. Consequently, simulator throughput, rather than raw qubit capacity, has become the primary bottleneck.

GPU, with their massive memory bandwidth and Single Instruction Multiple Data (SIMD) parallelism, are the natural accelerators for such workloads [5]. GPU-based libraries such as `cuStateVec` and `cuTensorNet` from the `cuQuantum` SDK exemplify this trend. However, the performance impact of different backends and noise channels remains underexplored —a gap this work aims to address [6].

1.1 Motivation

Simulating quantum circuits is a critical step in the development of quantum software. It allows researchers to test quantum algorithms, evaluate noise resistance, and validate circuit optimisations in the absence of accessible or sufficiently accurate quantum hardware. This is especially vital in the NISQ era, where most available processors are still noisy and limited in scale [7].

With the rise of GPU-accelerated backends (e.g. via NVIDIA's `cuQuantum` libraries) and frameworks such as `CUDA-Q`—and, more recently, GPU-enabled builds of `Qiskit Aer`—efficient simulation of quantum circuits has become more feasible for intermediate-size systems [5]. These tools promise high throughput and scalable memory performance, but comparative benchmarks—especially for noise-aware simulations—remain sparse [6].

This project is motivated by the desire to explore the practical performance of the two simulators. The focus lies on the Quantum Fourier Transform (QFT), as it is particularly well-suited for benchmarking, as it involves long-range entangling gates, making it highly sensitive to both hardware connectivity and noise [8, 9]. By implementing it in `Qiskit Aer` and `CUDA-Q`, the study aims to quantify the trade-offs between runtime, number of shots, and simulation accuracy under various configurations.

The study will begin with standard input states, such as the all-zero state $|0\rangle^{\otimes n}$ and Greenberger–Horne–Zeilinger (GHZ) states, and utilise fidelity-based metrics, including the L_2 norm, Frobenius norm, and state fidelity, to evaluate output accuracy [9]. These metrics will be introduced later in the theoretical framework.

1.2 Objectives

The primary objective of this PIC is to benchmark the performance and accuracy of modern quantum circuit simulators on a realistic workload. More specifically:

Implement and benchmark the QFT on both `Qiskit Aer` and `CUDA-Q`, running on CPU and GPU backends;

Evaluate simulator accuracy using state fidelity, Frobenius norm and L_2 distance for different input states (e.g., $|0\rangle^{\otimes n}$, GHZ);

Compare runtime and stability across simulators under ideal and noisy conditions, analysing circuit depth, qubit count and memory usage;

Explore the impact of multiple noise models on simulator throughput and output fidelity;

Ultimately, this work aims to contribute toward a better understanding of performance–accuracy trade-offs in GPU-accelerated quantum simulators.

1.3 Outline

This report is organised into the following chapters:

Chapter 2 — Background: Introduces the quantum computing formalism, with emphasis on qubits, quantum circuits and measurements.

Chapter 3 — Quantum Algorithms and Simulators: Describes the implementation of QFT benchmarks, then surveys the principal classical-simulation paradigms (state-vector, path-summation, Schrödinger–Feynman hybrid, and tensor-network methods) with a worked 4-qubit QFT example.

Chapter 4 — State-of-the-Art of Classical Quantum-Circuit Simulators: Compares leading software stacks—including `Qiskit Aer` and `CUDA-Q`—highlighting their back-ends, optimisation strategies, noise-model support, and hardware acceleration.

Chapter 5 — Preliminary Results and Thesis Proposal: Reports initial benchmarks on QFT circuits (ideal and noisy) across CPU/GPU back-ends, analyses accuracy and runtime trends, and sets out the five-phase work plan, methodology, expected deliverables, and timeline for the remainder of the master's project.

CHAPTER 2
Introduction of Quantum Computing

Quantum computation is formally grounded in a mathematical framework that extends core principles of quantum mechanics. Unlike classical computing, which operates on bits and deterministic logic gates, quantum computing exploits uniquely quantum phenomena such as superposition, entanglement, and probabilistic measurement.

This chapter introduces the fundamental principles of quantum mechanics that govern the behaviour of qubits and quantum circuits. It begins with an explanation of qubits, their properties, and how they differ from classical bits, followed by an overview of the operation of quantum circuits and their role in quantum algorithms.

2.1 Formalism Underpinning Quantum Computation

The foundational postulates of quantum mechanics presented below are reformulated to emphasise their application in quantum computing, particularly in qubit-based simulation and circuit dynamics.

Postulate 1 State Space

An isolated quantum register of n qubits is described by a unit state-vector $|\Psi\rangle$ in the 2^n-dimensional Hilbert space $\mathscr{H}_{2^n}$. Global phase is unobservable: $|\Psi\rangle \sim e^{i\phi}|\Psi\rangle$. For a single qubit ($n = 1$) the computational basis $\{|0\rangle, |1\rangle\}$ spans $\mathscr{H}_2$, and states are often visualised on the Bloch sphere.

Postulate 2 Observables as Hermitian Operators

Every measurable quantity corresponds to a Hermitian operator $\hat{O} = \sum_i o_i |o_i\rangle\langle o_i|$ whose real eigenvalues o_i are the only possible measurement outcomes. In quantum computing, the Pauli operators $\hat{X}, \hat{Y}, \hat{Z}$ and tensor products thereof are the most common observables.

Postulate 3 Measurement (Born Rule)

Measuring $\hat{O}$ on state $|\Psi\rangle$ yields outcome o_i with probability $\mathbb{P}(o_i \mid \Psi) = |\langle o_i | \Psi\rangle|^2$, after which the state collapses to $|o_i\rangle$.

Postulate 4 Time Evolution (Unitary Dynamics)

Between measurements, a closed quantum system evolves according to the Schrödinger equation $i\hbar \frac{d}{dt}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle$, whose solution is unitary $U(t) = e^{-i\hat{H}t/\hbar}$. The circuit model approximates this continuous evolution by a finite sequence of unitary gates $U = U_k \cdots U_2 U_1$ acting on the qubits.

2.2 Quantum Bits (Qubits)

In classical computing, a n-bit register can occupy exactly one of the 2^n configurations in $\mathbb{B}^n = \{0,1\}^n$. By contrast, a quantum register of n qubits lives in a 2^n-dimensional Hilbert space (**Postulate 1**), enabling superpositions across all basis states.

A single qubit is described by a normalised superposition of the computational basis $\{|0\rangle, |1\rangle\}$:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad \alpha, \beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1, \tag{2.1}$$

where α and β are probability amplitudes (**Postulate 3**).

More generally, an n-qubit system occupies a superposition of all 2^n basis states:

$$|\Psi\rangle = \sum_{x\in\{0,1\}^n} c_x|x\rangle, \quad \sum_x |c_x|^2 = 1, \tag{2.2}$$

with amplitudes $c_x \in \mathbb{C}$.

2.3 Quantum Measurement

A quantum measurement extracts classical information from a quantum state and is governed by the Born Rule (Postulate **Postulate 3**). If a system is in the generic state described by Equation (2.2) then a projective measurement in the computational basis $\{|x\rangle\}$ yields the outcome $|x\rangle$ with probability $|c_x|^2$. Immediately after obtaining outcome $|x\rangle$, the system collapses to that basis vector. Repeating the measurement on the same basis will deterministically return $|x\rangle$.

For instance, consider the state

$$|\psi\rangle \;=\; \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \tag{2.3}$$

commonly denoted $|+\rangle$. A computational-basis measurement yields:

$$\Pr(|0\rangle) = \left|\tfrac{1}{\sqrt{2}}\right|^2 = \tfrac{1}{2}, \quad \Pr(|1\rangle) = \left|\tfrac{1}{\sqrt{2}}\right|^2 = \tfrac{1}{2}. \tag{2.4}$$

After measuring $|+\rangle$, the post-measurement state collapses to $|0\rangle$ or $|1\rangle$ accordingly.

More generally, let A be a Hermitian operator with spectral decomposition $A = \sum_i a_i |a_i\rangle\langle a_i|$. If $|\alpha\rangle = \sum_i \langle a_i|\alpha\rangle\,|a_i\rangle$, then measuring A on state $|\alpha\rangle$ produces outcome a_i with probability $\Pr(a_i) = \left|\langle a_i|\alpha\rangle\right|^2$, and the post-measurement state becomes $|a_i\rangle$.

2.4 Bloch Sphere

An intuitive and widely used way to represent any single-qubit state—both pure and mixed—is through the Bloch Sphere (visualised in Figure 2.1). By **Postulate 1**, a pure state is a unit vector $|\psi\rangle = \alpha|0\rangle +$

$\beta\,|1\rangle$, with $|\alpha|^2 + |\beta|^2 = 1$. Up to the global phase, it can be written as:

$$|\psi\rangle \;=\; \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\,\sin\frac{\theta}{2}\,|1\rangle\,. \quad \theta \in [0,\pi],\; \varphi \in [0,2\pi), \tag{2.5}$$

The angles (θ, φ) map to Cartesian coordinates (x, y, z)=$(\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta)$ on the unit sphere (Figure 2.1). The poles correspond to $|0\rangle$ and $|1\rangle$; points the equator such as $|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$ and $\frac{1}{\sqrt{2}}(|0\rangle \pm i\,|1\rangle)$ lie along the x and y axes.

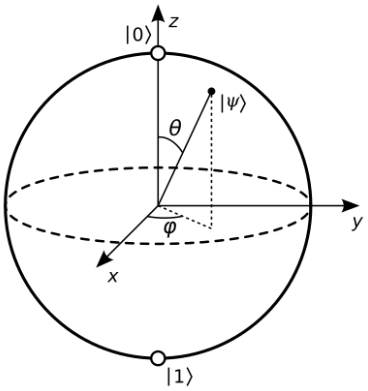


Figure 2.1: Bloch sphere: a geometrical representation of a two-level quantum system (adapted from Smite-Meister [10]).

Mixed states lie inside the sphere. Any single-qubit density matrix (see Section 2.4.1) can be expressed as:

$$\rho \;=\; \frac{1}{2}\Big(I + x\,\sigma_x + y\,\sigma_y + z\,\sigma_z\Big), \quad x^2 + y^2 + z^2 \le 1, \tag{2.6}$$

where (x, y, z) is the Bloch vector and σ_i are Pauli matrices. Pure states sit on the surface; the maximally mixed state $\frac{1}{2}I$ occupies the centre.

2.4.1 Mixed States and Density Operator

In practical quantum computing, systems often interact with their environment or undergo imperfect operations. In such cases, we cannot describe the state by a single state-vector $|\psi\rangle$, but instead use the density operator formalism, which captures both pure and mixed states.

A pure state is represented by a rank-one projector $\rho = |\psi\rangle\langle\psi|$. In contrast, a mixed state describes a statistical ensemble of pure states $\{|\psi_i\rangle, p_i\}$, and is expressed as:

$$\rho = \sum_i p_i\,|\psi_i\rangle\langle\psi_i|\,, \quad \text{with } \sum_i p_i = 1 \text{ and } p_i \ge 0. \tag{2.7}$$

The density operator ρ is satisfies three key properties: it is an Hermitian ($\rho^\dagger = \rho$), positive semidefinite ($\langle\psi|\rho|\psi\rangle \ge 0$ for all $|\psi\rangle$), and has unit trace ($\mathrm{Tr}(\rho) = 1$). A state is pure if and only if $\mathrm{Tr}(\rho^2) = 1$; otherwise, it is mixed ($\mathrm{Tr}(\rho^2) < 1$).

2.5 Entanglement and Nonlocality

Entanglement is a quantum phenomenon where the quantum state of a composite system cannot be factored into independent states for each individual subsystem. Instead, the system must be described by a collective state that accounts for the correlations between all subsystems. In quantum computing, entangled states are essential for achieving quantum speed-ups, enabling algorithms such as quantum teleportation, error correction, and certain subroutines in Shor's algorithm.

In the computational formalism of quantum computing, a bipartite pure state of two qubits A and B is described in the tensor product space $\mathscr{H}_A \otimes \mathscr{H}_B$. The state is called separable if it can be written as $\psi_{AB} = |\psi_A\rangle \otimes |\psi_B\rangle$. In contrast, the state is entangled if it cannot be factored into such product form. Entanglement implies that the qubits are in a state where their individual properties are no longer independent, even when they are physically separated.

A canonical example of an entangled state is the bell state:

$$|\Psi^+\rangle = \tfrac{1}{\sqrt{2}}(|01\rangle + |10\rangle). \tag{2.8}$$

The state is entangled because it cannot be expressed as a product of individual states of qubits A and B. Prior to measurement, Ψ^+ exists in an equal superposition of $|01\rangle$ and $|10\rangle$. When a measurement is made on qubit A, the state of qubit B is immediately determined, regardless of the physical separation between the qubits.

Entanglement is not just a theoretical concept, but a resource for quantum computing. It facilitates the creation of highly correlated states that are leveraged in various quantum algorithms and protocols. Examples include the four Bell states:

$$\begin{aligned} |\Phi^+\rangle &= \tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle), & |\Phi^-\rangle &= \tfrac{1}{\sqrt{2}}(|00\rangle - |11\rangle), \\ |\Psi^+\rangle &= \tfrac{1}{\sqrt{2}}(|01\rangle + |10\rangle), & |\Psi^-\rangle &= \tfrac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \end{aligned}$$

which form an orthonormal basis of maximally entangled two-qubit states. These states are integral to quantum information theory, enabling various quantum protocols such as superdense coding and quantum key distribution.

2.6 Quantum Circuits

Quantum circuits provide a diagrammatic framework for representing the unitary evolution and measurement of qubits over time (see Figure 2.2). Each horizontal line in the circuit represents a qubit, and time progresses from left to right. At time t = 0, each qubit is typically initialised in the computational state $|0\rangle$, unless otherwise specified. Quantum gates then act on one or more qubits, applying unitary transformations to modify their states. At the end of the circuit, a projective measurement may be performed

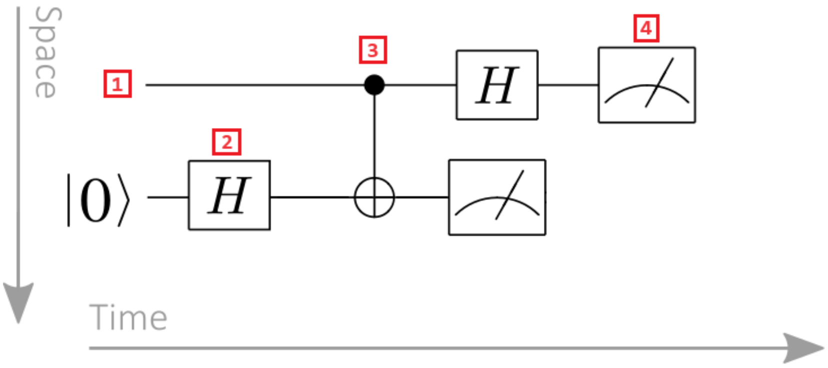


Figure 2.2: Quantum circuit illustrating key components: 1. **Qubit Registers:** Horizontal lines represent qubits $(0, 1, \ldots)$. 2. **Single-Qubit Gate:** Rectangular boxes for unitary operations (e.g., Hadamard gate H). 3. **Controlled Gate:** Control qubit marked with a circle; target qubit with a symbol (e.g., $\oplus$ for CNOT). 4. **Measurement:** Meter symbol for projective measurement, collapsing qubit to $|0\rangle$ or $|1\rangle$.

(see Section 2.3), collapsing each qubit to either $|0\rangle$ or $|1\rangle$ and yielding corresponding quantum bits.

2.6.1 Quantum Gates

Quantum gates are unitary operators that act on one or more qubits and form the fundamental components of quantum circuits. These gates enable the implementation of quantum algorithms, all while preserving the superposition and entanglement of quantum states. Unlike classical gates, quantum gates must adhere to the principles of unitarity, ensuring that quantum operations are reversible and that probability is preserved.

As unitary operators, quantum gates act on n-qubit states $|\psi\rangle$ and satisfy the condition $U^\dagger U = I$, ensuring that the quantum state norm is preserved. This is in line with **Postulate 4**, which states that quantum systems evolve unitarily between measurements. The evolution is described by a unitary operator $U(t) = e^{-i\hat{H}t/\hbar}$, and in quantum circuits, this continuous evolution is approximated this evolution by applying a finite sequence of unitary gates on the qubits.

A critical distinction of quantum mechanics is the impossibility of copying an arbitrary quantum state. This impossibility arises from the linearity of quantum theory, which prevents "cloning" an unknown qubit state. As a result, quantum information cannot be duplicated; instead, it is shared through entanglement or similar indirect mechanisms. This no-cloning theorem is a fundamental feature of quantum systems and underpins their uniqueness compared to classical systems [11, Section 5.1.1].

While quantum gates evolve the state of a system, measurement differs fundamentally. Measurement is described by projective operators (Section 2.3) and results in the irreversible collapse of the quantum state to a definite outcome. Since measurements cannot be undone, it is not classified as a quantum gate in the same sense as reversible unitaries.

Operator	Circuit Diagram(s)	Matrix
Pauli–X (X)	$-\boxed{X}-$ $\quad -\oplus-$	$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli–Y (Y)	$-\boxed{Y}-$	$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli–Z (Z)	$-\boxed{Z}-$	$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)	$-\boxed{H}-$	$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S or P)	$-\boxed{S}-$	$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)	$-\boxed{T}-$	$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$

Table 2.1: Examples of single-qubit gates.

Single-Qubit Gates Single-qubit gates act on the two-dimensional Hilbert space $\mathbb{C}^2$ and are represented by 2×2 unitary matrices. These gates are fundamental in quantum computations, enabling the manipulation of quantum states. Table 2.1 lists the most commonly used single-qubit gates and their matrix forms.

The Pauli gates X, Y, and Z implement π-rotation gates, each rotating a qubit around one of the coordinate axes (see Figure 2.3). The Hadamard gate H induces a π-rotation about the axis bisecting the x and z directions. It plays a crucial role in quantum algorithms by creating superpositions from basis states. For example, it maps $|0\rangle$ to $|+\rangle$ and $|1\rangle$ to $|-\rangle$.

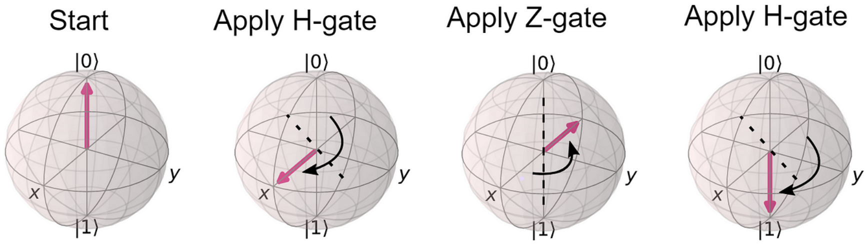


Figure 2.3: Pauli gates as π-rotations on the Bloch sphere: X (red), Y (green), and Z (blue).

Finally, the Phase gate S and the $\pi/8$ gate T induce purely z-axis rotations by fixed angles. Both leave $|0\rangle$ unchanged and multiply $|1\rangle$ by a phase i (for S) or $e^{i\pi/4}$ (for T), thus effecting a rotation about the z-axis without altering the Bloch vector's x- or y-components.

Multi-Qubit Gates Multi-qubit gates are crucial components of quantum circuits, enabling conditional operations and entanglement. Table 2.2 presents examples of commonly used multi-qubit gates. These gates allow for quantum circuits to perform operations such as AND, OR, and other classical logic gates

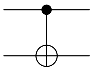
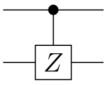
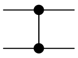
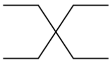
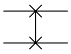
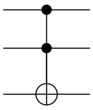
Operator	Circuit Diagram(s)	Matrix
Controlled NOT (CNOT, CX)		$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled–Z (CZ)	 	$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP	 	$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX)		CCNOT=$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Table 2.2: Examples of multi-qubit gates

that single-qubit gates alone cannot achieve.

Among the earliest and most important multi-qubit gates is the Controlled-NOT (CNOT) gate [12]. The CNOT gate operates on two qubits: a control qubit and a target qubit. It flips the state of the target qubit only when the control qubit is in the state $|1\rangle$. This gate is foundational for quantum computing, enabling the creation of entangled states, such as Bell states, which are essential for quantum algorithms like Shor's factoring algorithm and Grover's search algorithm.

Formally, for computational basis states $|c, t\rangle$, with $c, t \in \{0, 1\}$, the action of CNOT gate is:

$$\text{CNOT } |c, t\rangle = |c,\, t \oplus c\rangle\,, \tag{2.9}$$

By implementing conditional state flipping, the CNOT gate plays a pivotal role in quantum algorithms, making it a cornerstone of multi-qubit quantum circuits.

Method	Best For
Schrödinger (State-vector)	Exact simulations, ≤ 30 qubits
Feynman (Path-summation)	Shallow circuits
Schrödinger–Feynman (Hybrid)	Intermediate depth
Tensor Networks (MPS, etc.)	Low-entanglement, large systems

Table 2.3: Simulation strategies and their ideal use cases.

CHAPTER 3
Quantum Algorithms and Simulators

From a computational complexity perspective, simulating quantum systems is inherently demanding. The state of a n-qubit is represented by a complex vector of dimension 2^n, with unitary evolution acting via $2^n \times 2^n$ matrices. Storing and manipulating these exponentially large objects impose severe demands on classical memory and computational resources, especially under naive state-vector propagation [13].

This chapter introduces the core quantum algorithm that serves as a benchmark in this work—the QFT—and presents the main classical techniques for simulating such circuits. Emphasis is placed on the computational complexity of each method, their practical performance, and their deployment on modern hardware platforms, especially those leveraging GPU acceleration. The chapter also includes a detailed worked example of the QFT on a 4-qubit register, illustrating both the theoretical formulation and the simulation strategy.

3.1 The Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is a fundamental subroutine in quantum computing, notably it is a key component of Shor's algorithm for integer factorization [14], quantum phase estimation and algorithms addressing the Abelian Hidden Subgroup Problem (HSP). As the quantum analogue of the classical Discrete Fourier Transform (DFT), the QFT efficiently extracts periodicity and spectral information from quantum states, operating on both product and entangled states. Its ability to perform this task exponentially faster than classical Fourier transforms makes it indispensable for some quantum algorithms.

In classical computation, the DFT acts on a N-component complex vector $(x_0, x_1, \ldots, x_{N-1})$, where $N = 2^n$ and n is the number of input bits. It produces the output vector $(y_0, y_1, \ldots, y_{N-1})$, given by:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i jk/N}, \quad \text{for } k = 0, \ldots, N-1. \tag{3.1}$$

In the quantum setting, analogously to the classical DFT, the QFT acts as a unitary transformation on an orthonormal basis $|0\rangle, \ldots, |N-1\rangle$, where $N = 2^n$ and n is the number of qubits. The transformation is defined by its action on computational basis states:

$$|j\rangle \xrightarrow{\text{QFT}} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i\, jk/N} \, |k\rangle\,, \qquad 0 \le j < N. \tag{3.2}$$

By linearity, the QFT extends to arbitrary quantum states. For a general state $|\psi\rangle = \sum_{j=0}^{N-1} x_j \, |j\rangle$, the transformation yields:

$$|\psi\rangle \xrightarrow{\text{QFT}} \sum_{k=0}^{N-1} y_k \, |k\rangle \tag{3.3}$$

where the amplitudes y_k correspond to the DFT of the amplitudes x_j. Thus, the QFT implements the DFT on a quantum register up to a normalisation factor of $1/\sqrt{N}$, and does so in a unitary manner.

Although this property is not immediately evident from its definition in Equation (3.2), it is indeed a unitary transformation. As a unitary operator, the QFT preserves inner products and guarantees reversibility—both essential requirements for implementation in quantum circuits (discussed in Section 2.6). To explicitly demonstrate the unitarity and to develop further intuition, consider the non-trivial case with $N = 2$, i.e., a single qubit:

$$|0\rangle \xrightarrow{\text{QFT}} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle\,, \qquad\qquad |1\rangle \xrightarrow{\text{QFT}} \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle\,. \tag{3.4}$$

These output states correspond to those obtained by applying the Hadamard gate H to the computational basis. Hence, in the single-qubit case, the QFT coincides exactly with the Hadamard transformation $U^{(1)}_{\text{QFT}} = H$. This correspondence both confirms the unitarity of the QFT and connects its action to the geometric picture of Bloch-sphere rotations discussed earlier in Section 2.6.1.

3.1.1 Circuit Implementation of QFT

To implement the QFT circuit, it needs to undergo a Hadamard transformation gate on each qubit, followed by a series of controlled phase shifts. These rotations have relative phases depending on the number of the more significant qubits.

Let us recall that the Hadamard gate H, from Section 2.6.1 acts as:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{3.5}$$

which creates a superposition by mapping computational basis states $|0\rangle$ and $|1\rangle$ to $|+\rangle$ to $|-\rangle$, respectively.

Additionally, that the controlled-R_k rotation gate is defined by:

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}. \tag{3.6}$$

These R_k gates are applied conditionally based on the states of more significant qubits, introducing

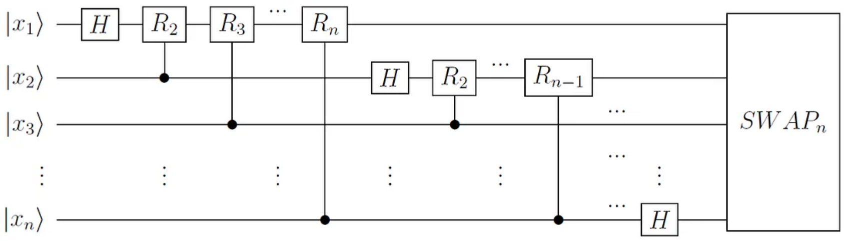


Figure 3.1: Schematic n-qubit QFT: each qubit undergoes a Hadamard H followed by a cascade of controlled rotations R_k from more-significant qubits; a final bit-reversal layer (SWAP chain) restores natural order. [15]

binary-weighted phases to the quantum state.

Each qubit undergoes a Hadamard transformation followed by a series of controlled-R_k rotations. For example, the first qubit, known as the least-significant qubit (Least Significant Qubit (LSQb)), is transformed as follows:

$$|x_0\rangle \mapsto \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\,0.x_0}\,|1\rangle\right), \tag{3.7}$$

where x_0 is the binary expansion of the input. Subsequent controlled-R_k gates apply additional phases based on the states of the higher-order qubits. This process continues for each qubit, with the transformation being applied in the order of qubit significance.

The transformation of the first qubit $|x_0\rangle$ is then followed by subsequent qubits in the system, each undergoing the same transformation sequence. After each qubit has undergone a Hadamard transformation and a set of controlled-R_k rotations, the resulting state is a superposition of all possible states with the appropriate phase shifts.

The qubit order in the QFT circuit naturally reverses as part of the transformation. To restore the original qubit order, a sequence of SWAP gates is applied at the end of the circuit. Specifically, for a system of n qubits, $\lfloor n/2 \rfloor$ SWAP gates are sufficient to reverse the qubit order.

Figure 3.1 presents the complete circuit-level implementation of the n-qubit QFT, incorporating the final step of qubit reordering via SWAP gates.

3.1.2 Worked Example: QFT on 4 Qubits

Consider the QFT acting on 4 qubits. For the input $|0000\rangle$, the output is

$$U^{(4)}_{\text{QFT}}\,|0000\rangle = \frac{1}{\sqrt{16}}\sum_{k=0}^{15}|k\rangle, \tag{3.8}$$

a uniform superposition over all 16 basis states, yielding $P(k) = 1/16$ by Born's rule.

For the entangled input, $|\mathrm{GHZ}_4\rangle = \frac{1}{\sqrt{2}}(|0000\rangle + |1111\rangle)$ linearity gives

$$U_{\mathsf{QFT}}^{(4)}\,|\mathrm{GHZ}_4\rangle = \frac{1}{\sqrt{32}}\sum_{k=0}^{15}\left(1 + e^{2\pi i\,15k/16}\right)|k\rangle\,. \tag{3.9}$$

This leads to an interference pattern where all $|k\rangle$ may appear, but with amplitudes governed by phase interference. The probability of outcome $|k\rangle$ is

$$P(k) = \frac{1}{32}\left|1 + e^{2\pi i\,15k/16}\right|^2. \tag{3.10}$$

In contrast to the uniform output of $|0000\rangle$, the GHZ case yields a symmetric but non-uniform distribution, with constructive and destructive interference across the spectrum. Simulated results for both inputs are compared to theory in Section 5.3.1.

3.2 Clifford Circuits and the Gottesman–Knill Theorem

As discussed in Section 2.6.1, circuits built only from Clifford gates—Hadamard (H), phase (S), and controlled-NOT (CNOT)—are not universal for quantum computation, but they can be simulated efficiently on a classical computer. This is formalised by the Gottesman–Knill theorem:

Theorem 1 (Gottesman–Knill [16])**.** *Any Clifford circuit acting on the state* $|0\rangle^{\otimes n}$*, followed by measurements in the computational basis, can be classically simulated in polynomial time, including the computation of exact probabilities and amplitudes.*

This result is made possible by the stabiliser formalism, where the quantum state is described by a set of stabiliser operators rather than a full state-vector. Efficient algorithms, such as the tableau method [17], enable simulation of large Clifford circuits with thousands of qubits.

While Clifford circuits are not universal, they are still highly relevant in quantum computing. Many Quantum Error Correction (QEC) protocols and parts of quantum algorithms use Clifford operations. Moreover, adding a few non-Clifford elements (like T gates or magic states) to mostly-Clifford circuits allows simulation techniques to still exploit the stabiliser structure for improved efficiency [18].

3.3 Strong vs. Weak Simulation

As discussed in Section 2.3, quantum measurements are inherently probabilistic. A projective measurement in the computational basis returns the basis state $|x\rangle$ with probability $\Pr(x) = |\langle x|\psi\rangle|^2$, where $|\psi\rangle$ is the system's state immediately prior to measurement. Thus, repeated measurements on identically prepared states yield outcomes sampled from this fixed probability distribution.

Depending on the level of detail required, a classical simulator may either compute the full quantum state, enabling exact evaluation of measurement probabilities for arbitrary observables, or generate samples that replicate the circuit's output distribution. These two modes of operation correspond to strong and weak simulation, respectively.

Strong Simulation computes the full quantum state or specific outcome probabilities with high precision. This includes retrieving amplitudes $\langle x|U|0^n\rangle$ or exact probabilities $|\langle x|\psi\rangle|^2$, making it useful for debugging, fidelity estimation, and theoretical analysis. However, it requires exponential memory and runtime, becoming impractical beyond ~ 30–40 qubits except for restricted circuit families (e.g., Clifford circuits; see Section 3.2) [19].

Formally, let a n-qubit circuit implement the unitary U and end with a computational-basis measurement of the first qubit. A strong simulator must, on input U_n be able to return

$$\Pr(0) = \mathrm{Tr}\big[\,|0\rangle\langle 0|\;U\;|0^n\rangle\langle 0^n|\;U^\dagger\big] \tag{3.11}$$

and, more generally, any amplitude $\langle x|\,U\,|0^n\rangle$ to within an arbitrarily small additive error.

For generic circuits, this task is #P-hard[1], but special families—most notably Clifford (stabiliser) circuits—admit efficient classical algorithms via the Gottesman–Knill theorem (see Section 3.2) [19].

Weak Simulation requires only the ability to sample classical outcomes with the correct probabilities, without ever storing the full quantum state. More precisely, let a n-qubit circuit implement the unitary U and end with a computational-basis measurement of the first qubit. A weak simulator, on input (U, n),

$$\text{outputs } 0 \text{ with probability } \mathrm{Tr}\big[|0\rangle\langle 0|\;U\,|0^n\rangle\langle 0^n|\,U^\dagger\big], \quad \text{and } 1 \text{ otherwise.} \tag{3.12}$$

Equivalently, for circuits that measure all qubits, a weak simulator produces bitstrings $x \in \{0,1\}^n$ whose empirical frequencies converge to the true distribution $\Pr(x) = |\langle x|U|0^n\rangle|^2$.

Because it avoids explicit state-vector construction, weak simulation can exploit techniques such as tensor-network contraction, path-integral Monte Carlo, or amplitude partitioning to achieve much lower runtime and memory overhead—albeit at the cost of forfeiting exact amplitude queries and other fine-grained information about the state [19, 20].

[1] It implies that exactly computing output probabilities—or even a single amplitude—is at least as hard as the most difficult counting problems, and is believed to be intractable on classical hardware.

3.4 Accuracy Metrics

The accuracy of quantum circuit simulation can be quantified through several well-established metrics that capture the degree of agreement between the simulated output and a theoretical reference. These are particularly important in the presence of noise, where deviations from the ideal evolution must be rigorously assessed.

One such metric is state fidelity, which evaluates the overlap between two quantum states. For density matrices ρ and σ, fidelity is defined as

$$F(\rho, \sigma) = \left(\mathrm{Tr} \sqrt{\sqrt{\rho} \sigma \sqrt{\rho}} \right)^2 . \tag{3.13}$$

In cases where one state is pure, for example $\rho = |\psi\rangle \langle\psi|$, this expression simplifies to $F(\rho, \sigma) = \langle\psi| \sigma |\psi\rangle$, yielding a direct measure of state overlap. Fidelity takes values in the interval $[0, 1]$, where 1 indicates perfect agreement.

A second important metric is the Frobenius norm, used to measure the distance between two density matrices. This is defined as:

$$\|\rho - \sigma\|_F = \sqrt{\mathrm{Tr}\left[(\rho - \sigma)^\dagger (\rho - \sigma)\right]}, \tag{3.14}$$

and provides a matrix-level norm that reflects how much the simulated state deviates from the reference in terms of overall amplitude structure.

Finally, the L_2 distance quantifies the difference between the simulated $\mathbf{p}_{\mathrm{sim}}$ and ideal $\mathbf{p}_{\mathrm{ideal}}$ outcome distributions:

$$L_2 = \|\mathbf{p}_{\mathrm{sim}} - \mathbf{p}_{\mathrm{ideal}}\|_2 = \sqrt{\sum_i \left(p_i^{\mathrm{sim}} - p_i^{\mathrm{ideal}}\right)^2}. \tag{3.15}$$

It is sensitive to shifts in distribution weight, making it useful for assessing the statistical impact of noise and sampling errors. However, it is less effective at detecting fine-grained state-level discrepancies, especially when the overall distributions remain similar. While the L_2 norm offers valuable insight into output distributions, it may fail to capture deeper structural differences between quantum states.

CHAPTER 4

State-of-the-Art of Classical Quantum Circuit Simulators

As quantum hardware continues to advance, accurate and scalable classical simulation remains essential for validating circuits, testing noise models, and benchmarking quantum algorithms. This chapter reviews the principal simulation approaches used in classical emulation of quantum circuits, comparing their computational trade-offs, strengths, and limitations. Particular attention is given to the tools used in this work—Qiskit Aer and CUDA-Q—and their support for these simulation strategies.

4.1 Quantum Simulation Approaches

This section explores three primary techniques used in the classical simulation of quantum circuits: the Schrödinger (state-vector) method, Feynman (path-summation) simulation, and the Hybrid simulation approach. Each method has its own trade-offs in terms of time and space complexity, and their applicability depends on the quantum circuit being simulated.

4.1.1 Feynman (Path-Summation) Simulation

The Feynman path-summation method (also called a sum-over-paths simulation) avoids storing the full 2^n-dimensional state-vector. Instead, the simulator explores all possible computational paths through the circuit, branching at each multi-qubit gate and accumulating the contribution of each path to the final amplitudes. In other words, every two-qubit (or entangling) gate introduces a “decision point” where the simulation splits into different cases (paths) depending on the possible basis states of the qubits involved. By securely following each path and summing their complex amplitudes at the end, one can compute the resulting quantum state or specific output probabilities. This approach reduces memory requirements—at any point, one only needs to track the state along a single path (requiring space on the order of $\mathcal{O}(n+m)$ for n qubits and m gates, rather than 2^n). However, it typically incurs a huge time cost: the number of paths grows exponentially with the number of branching points (entangling gates), making runtime the bottleneck [21]. In the worst case of a fully entangling circuit, the time complexity is on the order of $\mathcal{O}(4^m)$ (exponential in the gate count) or worse.

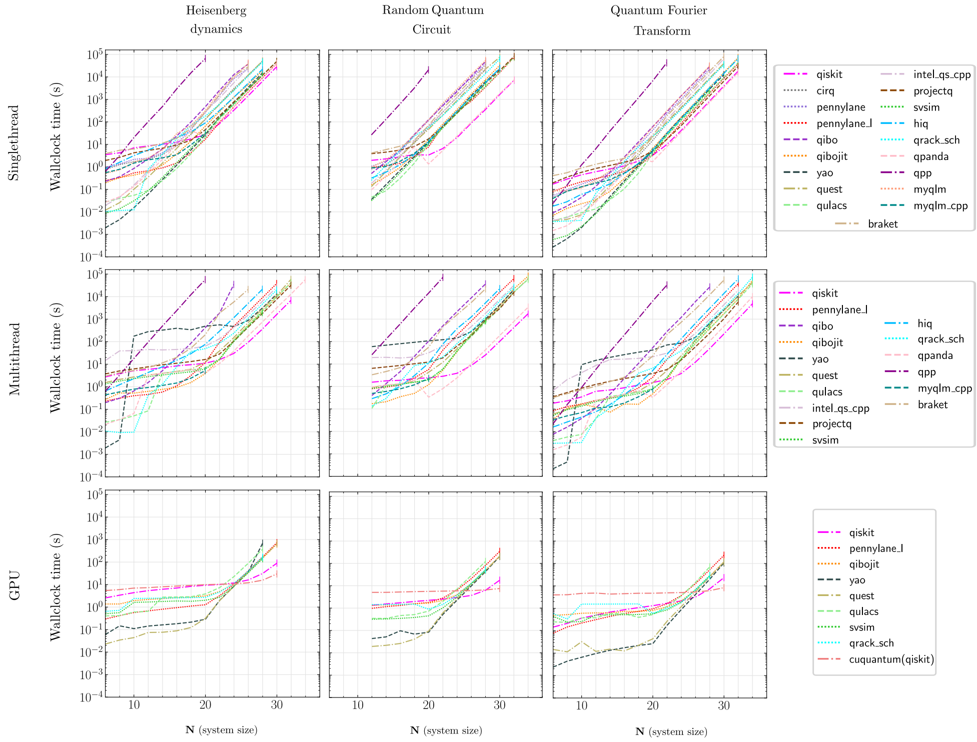


Figure 4.1: Performance comparison of simulation packages supporting double precision across different hardware architectures [23]

4.1.2 Schrödinger (State-Vector) Simulation

In the Schrödinger method, a n-qubit system is represented by its full state-vector in a 2^n-dimensional Hilbert space. This is a strong simulation, as it stores and updates the entire state-vector after each gate application, which allows for exact measurement probabilities and amplitudes [13]. However, memory quickly becomes a bottleneck–storing a general state of $n \approx 50$ qubits (with double precision) would require on the order of 2^{50} complex amplitudes (tens of petabytes), an impractical demand on any hardware [22].

Despite these limitations, modern-day state-vector simulators mitigate some of these issues with heavy optimisations and parallelism. Multi-core CPUs and GPUs substantially accelerate gate operations on the large state-vector. Indeed, GPU or multi-threading can yield significant speed-ups over single-core implementations.

Figure 4.1, from Gangapuram et al. [23], compares wallclock runtimes across multiple simulation packages for three canonical quantum workloads: Heisenberg dynamics, Random Quantum Circuits, and the QFT. The benchmarks include both purely state-vector simulators (e.g., `qibo`, `yao`, `quest`) and tools that may employ hybrid strategies (e.g., `qrack_sch`, `cuquantum(qiskit)`), though all simulate full amplitudes. As shown, GPU-based simulators introduce a significant speed-up over single-core implementations; here, specifically enabling the GPU and multi-threading leads to an order of magnitude

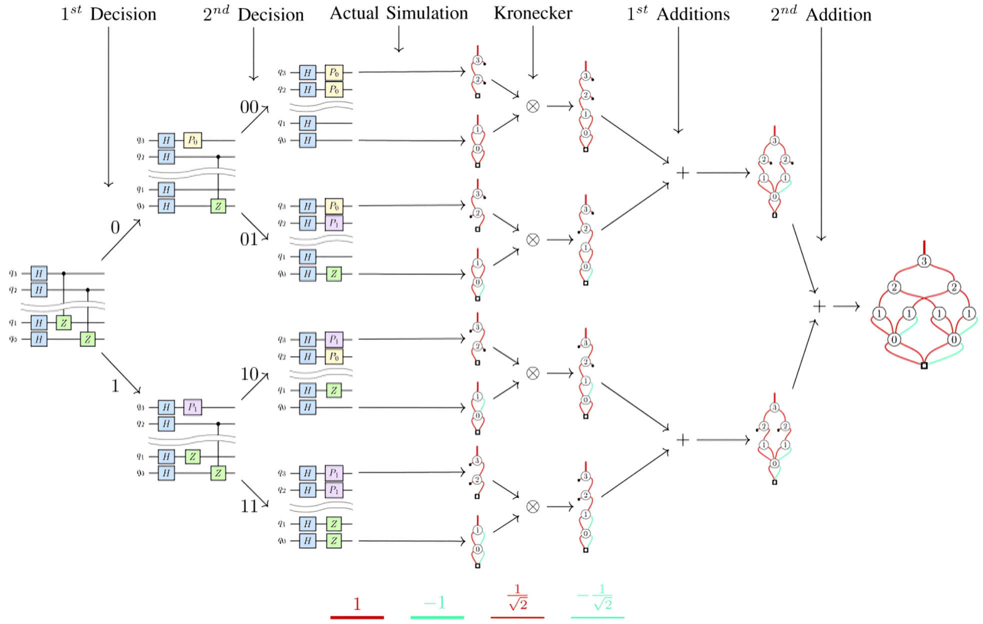


Figure 4.2: One complete pass of the Schrödinger–Feynman (Schrödinger–Feynman (SF)) hybrid algorithm on a four-qubit toy circuit. The diagram reads from left to right: (1)–(2) Decisions — each cut qubit is fixed to 0 or 1, producing the 2^k assignments $00, 01, 10, 11$; Actual Simulation — every branch is now a smaller stand-alone circuit simulated with an ordinary Schrödinger state-vector engine (shown as a quantum decision diagram, QDD, whose red/teal edges encode $\{+1, -1, \pm 1/\sqrt{2}\}$ phase factors); Kronecker — because the circuit was sliced horizontally, the top- and bottom-half state-vectors are tensored together; Additions — branches that differ only in the second decision are first added, and the resulting two QDDs are then added once more to yield the full n-qubit state on the right. The procedure uses $2^{n_1} + 2^{n_2}$ memory (two state-vectors) but repeats the simulation 2^k times, thereby trading exponential time for an exponential memory saving.

faster simulation, most noticeable once the system size increases. However, it's also noticeable that exponential scaling eventually dominates, as at around 26 qubits for the GPUs and around 20 qubits for multithreading, depending on the simulator, these enter the steep exponential time regime [23]. So while these simulators may try to optimise to lower the prefactor of the exponential scaling curve, allowing to reach a few more qubits or deeper circuits within a given time limit, for example, CUDA-level GPU optimisations (e.g., kernel fusion, register-level ops), they must contend with exponential memory and time demands, as it doesn't alter the underlying $\mathcal{O}(2^n)$ memory scaling.

Therefore, state-vector simulation provides high-fidelity, exact results and relatively straightforward implementation, but its poor memory scalability is the limiting factor. This motivates alternative approaches that sacrifice generality or exactness to simulate larger quantum circuits.

4.1.3 Schrödinger–Feynman (Hybrid) Simulation

The SF hybrid method strikes a balance between the time and space complexities of the Schrödinger and Feynman approaches. This method, represented in Figure 4.2, partitions the quantum circuit into two (or more) parts such that each part can be simulated by a Schrödinger (state-vector) method within available memory, and the interface between parts is handled by Feynman-style summation over possible intermediate state [21]. Each choice of values for the cut qubits (i.e., those that connect the two partitions) yields two independent sub-circuits – one for the first part and one for the second – which can be simulated like smaller standalone circuits. The results are then combined by summing over all 2^k combinations (hence Feynman summation over the cut).

The hybrid method allows the simulation to be distributed, with each simulation branch computed independently. Its time complexity is $T_{\mathrm{SF}} = \mathcal{O}(n \cdot 2^{n-k}(2d)^{k+1})$ and its space complexity $S_{\mathrm{SF}} = \Theta(2^{n-k} + n)$, where n is the number of qubits, k is the block size, and d is the circuit depth.

Looking at the benchmark introduced earlier, in Fig. 4.1, the performance gap between the hybrid SF back-ends and the best pure state-vector simulators is almost imperceptible. This is mainly because the state-vector engines already incorporate an arsenal of optimisations—gate fusion, just-in-time (JIT) kernel generation, batched shots, and aggressive cache/GPU tiling—which mask the memory overhead that SF is designed to avoid. More importantly, the problem sizes chosen for the study ($\approx 5\breve{}30$ qubits and fewer than $\mathcal{O}(10^2)$ gates) do not yet push the simulation into the regime where SF's space–time trade-off becomes decisive.

The benefits of the SF method become more apparent in scenarios where memory is the dominant constraint. Specifically, SF gains traction when the circuit width approaches the point where a 2^n–amplitude array no longer fits in device or host RAM (typically $n \gtrsim 35\breve{}40$ on commodity hardware), or the depth and entanglement are high enough that per-gate state-vector updates become prohibitively slow. For the comparatively modest circuits in the benchmark, highly-optimised state-vector back-ends often equal—or even surpass—the SF runtimes.

4.1.4 Tensor Network Approaches to Quantum Simulation

Tensor Network (TN) simulators take a very different approach by exploiting the often-limited entanglement structure of many circuits. Instead of storing a 2^n amplitudes explicitly, a TN represents the quantum state (or the circuit's action) as a connected network of many smaller tensors. This can dramatically reduce memory and computation requirements when the quantum state has some structure (e.g. obeying an area-law entanglement scaling) [22]. A prime example is using a Matrix Product State (MPS) representation for circuits that primarily generate one-dimensional nearest-neighbour entanglement—in such cases the required bond dimension (a measure of entanglement complexity) might remain small even as n grows, so that the memory scales only linearly or polynomially with n instead of 2^n.

In practice, this means TN methods can reach qubit counts inaccessible to brute-force state–vector simulators. For example, Qibo's tensor-network backend (`Qibotn`) simulated a 400-qubit variational circuit, whereas the full state–vector implementation (`Qibojit`) ran out of memory near 40 qubits (Fig. 4.3) [22].

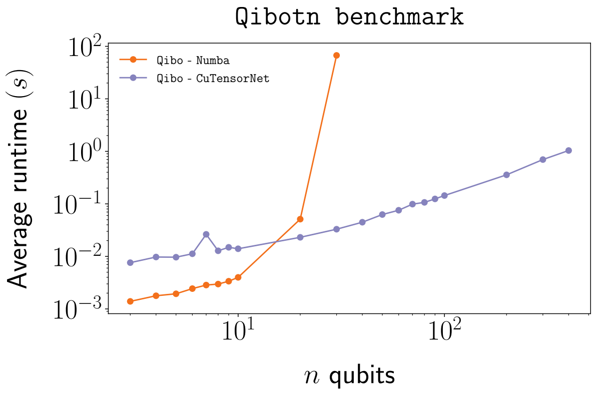


Figure 4.3: Total simulation time for a variational circuit: `Qibojit` (state–vector, AMD EPYC 7713) vs. `Qibotn` (tensor–network, NVIDIA A100) [22].

The `Qibotn` runtime flattens as n increases, reflecting that adding qubits with only local entanglement incurs only polynomial overhead.

These gains depend critically on entanglement structure. To remain tractable, TN simulators truncate small singular values or cap χ, introducing a controllable truncation error. Recovering full fidelity would require unbounded χ growth—restoring exponential memory and runtime. Highly entangling circuits (e.g. deep Random Quantum Circuits (RQCs)) drive χ exponentially, forcing either large truncation errors or exponential slowdown. Furthermore, contraction cost grows with χ and the network's treewidth; for two-dimensional layouts, this can still scale exponentially in the linear grid size even at small depth.

State–of–the–art TN libraries (e.g. NVIDIA's `cuTensorNet`) map large tensor contractions onto GPUs and, in multi-node setups, distribute tensor slices across a cluster [24]. This high–performance support parallels that for state–vector simulators and was essential in early "quantum supremacy" demonstrations (e.g. Google's 53-qubit experiment in 2019 [24]) where tensor–network contraction on supercomputers outperformed naive vector updates.

In summary, tensor-network simulation offers exceptional qubit count scalability (hundreds of qubits) for circuits with limited, structured entanglement. Its adjustable bond dimension χ trades fidelity for efficiency, but when entanglement grows beyond area–law behaviour, χ must grow accordingly—eroding the resource advantage and, in the worst case, matching the exponential cost of full state–vector methods.

4.2 Simulation Tools and Backends

This section presents the principal simulation frameworks utilised in this work: `Qiskit Aer` developed by IBM, and `CUDA-Q`, developed by NVIDIA.

4.2.1 Qiskit Aer

`Qiskit Aer` constitutes the high-performance simulation stack of IBM's open-source Qiskit framework. Written in C++ with Python bindings, Aer inter-operates seamlessly with Qiskit's circuit–construction and transpilation tool-chain, thereby allowing users to compile, optimise, and benchmark the very same circuits that can later be executed on IBM Quantum hardware [25].

This simulator exposes several specialised backends, each tailored to a particular algorithmic regime:

Statevector A dense Schrödinger-style engine that propagates pure states of dimension 2^n exactly. It is well suited to noise-free circuits up to the middle-scale regime ($n \lesssim 30$ on a single GPU/CPU socket).

Density-matrix A dense mixed-state simulator that evolves the $2^n \times 2^n$ density operator and therefore accommodates arbitrary Kraus-map noise. While memory intensive, this approach enables numerically exact studies of decoherence and cross-talk.

Clifford (stabiliser) A tableau-based backend that exploits the Gottesman–Knill theorem to simulate Clifford circuits in polynomial time and space. When all gates and noise channels are Clifford-preserving, circuits with thousands of qubits become tractable (cf. Section 3.2).

Extended-stabiliser An approximate method for Clifford + T circuits that decomposes the state into a superposition of stabiliser states, trading accuracy for scalability as the non-Clifford (T) count grows.

Matrix-product-state (MPS) A tensor-network backend that represents the wave-function as a one-dimensional MPS. Exact contraction is the default, but optional bond-dimension truncation enables controlled approximations for circuits with limited entanglement.

Tensor-network (cuTensorNet) A GPU-only backend based on NVIDIA's cuTensorNet library; it performs general tensor contractions for both statevector and density-matrix modes, and excels on shallow, locally connected circuits.

Aer supports shared-memory parallelism via `OpenMP`, GPU acceleration through `CUDA`, and distributed execution with Message Passing Interface (MPI)-based circuit slicing [26, 27]. A rich catalogue of parametrisable noise models—including Pauli channels, amplitude-damping, read-out errors, and device-calibrated mixtures—renders Aer particularly suitable for realistic NISQ-era performance studies [28].

4.2.2 CUDAQ

`CUDA-Q` is NVIDIA's framework for quantum simulation and hybrid quantum-classical workflows. It is designed for GPU execution and interfaces closely with the `cuQuantum` SDK, which provides high-performance libraries for quantum simulation primitives [29]. The framework is implemented in C++ and exposes a Python interface, enabling seamless prototyping and integration into quantum software pipelines [30]. It includes two simulation methods:

Statevector simulation (`cuStateVec`). Supports high-performance simulation of pure states via gate fusion, stream concurrency, and GPU slicing techniques.
TN simulation (`cuTensorNet`). Contract-based simulation that enables scalability to large qubit counts under restricted entanglement.

`CUDA-Q` provides support for multi-GPU execution via MPI and offers greater control over hardware-level optimisations compared to high-level frameworks. However, it does not currently support built-in noise modelling; external mechanisms such as sampling over noisy trajectories must be used to approximate decoherence [29].

4.3 Noise Simulation

Until this point, only ideal quantum circuits have been considered. However, current quantum hardware operates in the NISQ regime, where imperfections such as faulty gates, decoherence, and cross-talk introduce unavoidable noise [7]. Accurately modelling these effects is essential for realistic benchmarking and the development of noise-mitigation algorithms.

Density-matrix propagation The quantum state is represented by a density operator, which evolves under quantum channels that model noise processes such as amplitude damping, dephasing, and depolarisation. A generic channel $\mathcal{E}$ acts via

$$\rho \;\mapsto\; \mathcal{E}(\rho) \;=\; \sum_k E_k\, \rho\, E_k^\dagger, \tag{4.1}$$

thereby computing ensemble statistics in a single pass with exact accuracy. This strong simulation requires storing a $2^n \times 2^n$ matrix, scaling as $\mathcal{O}(4^n)$, which rapidly exceeds available memory beyond approximately 10–12 qubits [31].
Some simulation strategies extend naturally to this regime: for example, tensor-based approaches (cf. Section 4.1.4) can be adapted to evolve density matrices when the circuit's entanglement remains bounded [32]. By contrast, density-matrix simulation using Schrödinger or Feynman methods incurs significant memory overhead, making them less practical for noisy circuit evolution.

Stochastic noise-gate sampling (pure-state trajectories) In the Monte-Carlo wave-function or quantum jump picture, the system is propagated as a state vector. At each noisy location, a random Kraus operator E_k is sampled with probability $\Pr(k) = \mathrm{Tr}[E_k\, \rho\, E_k^\dagger]$, and the state-vector is renormalised to $|\psi\rangle \propto E_k\, |\psi\rangle$ [33]. Averaging over N_{traj} trajectories recovers the mixed-state result to statistical precision $\sim 1/\sqrt{N_{\text{traj}}}$. Memory usage remains $\mathcal{O}(2^n)$ per trajectory, making this strategy attractive when full density-matrix storage is infeasible.

CHAPTER 5

Preliminary Results and Thesis Proposal

5.1 Thesis Proposal

As previously discussed, the primary goal of this dissertation is to evaluate the performance of modern classical simulators for quantum circuits. Chapter 4 reviewed the major simulation techniques and surveyed existing benchmarks, most of which focus on state-vector simulators. Recent studies have also highlighted promising results from Tensor Network simulators, particularly for circuits with limited entanglement. However, the current benchmarking landscape remains narrow in scope, with limited coverage of diverse algorithms, noise modelling, and broader performance metrics.

This thesis aims to extend existing benchmarking efforts by incorporating a wider set of quantum algorithms, exploring additional simulators, and evaluating performance across a broader set of indicators, including noise resilience.

5.2 Methodology and Expected Results

Building on the simulator survey presented in Chapter 4, this section describes how the benchmarking framework will be executed and what outcomes are anticipated.

The benchmarking harness will incorporate some of the quantum circuit simulators, such as `Qiskit Aer`, `CUDA-Q`, `PyQrack` and `Qibo` simulators, with a focus on leveraging `cuQuantum`'s GPU acceleration, `Qiskit`'s integrated noise models and `CUDA-Q` libraries (`cuStateVec` and `cuTensorNet`). Benchmark circuits will include the QFT and simplified subroutines of Shor's algorithm (modular extension and phase estimation in small integers), simulated across a scalable range of qubit counts, up to the point where each simulator reaches its resource limits.

Each circuit will be compiled into the native gate set and connectivity model required by the target backend. Whenever supported, two variants will be generated per simulator: a noiseless version and a noisy version, with depolarising, bit-flip, amplitude-damping, and phase-damping channels interleaved at specified error rates. As simulation complexity increases with qubit count and noise, the study will also test the ability of simulators to scale.

Accuracy will be quantified using three complementary metrics. First, the state fidelity will compare the simulated QFT outputs against their analytically known counterparts. Second, the Frobenius norm will measure deviations between simulated and reference density matrices under noise. Third, for full-circuit sampling, the L_2 distance between empirical and ideal output probability distributions will be computed. These accuracy measures will be plotted as functions of qubit count and circuit depth, together with performance curves (run time and memory usage), to pinpoint the onset of each simulator's exponential "wall."

Given the limitations of each simulator technique stated in Chapter 4, it is expected that state-vector engines will maintain exact fidelity up to moderate qubit counts $(20 - 30)$, but will collapse in run-time or memory by $40 - 50$ qubits; tensor-network methods should outperform once entanglement remains sufficiently low and hybrid or branch-pruning approaches (as in `PyQrack`) will extend the feasible qubit range at the cost of approximate fidelity. Noisy simulations should reveal differing resilience: backends using pure-state trajectories may degrade more smoothly than full density-matrix engines.

Finally, depending on progress and available time, QEC codes will be embedded into circuits and their simulation cost profiled and methods such as reinforcement learning or neural network–based heuristics may be used to optimise circuit depth or noise robustness.

Together, these experiments will produce:

A systematic, multi-backend performance and accuracy profile for both ideal and noisy QFT and Shor subroutines;
an explicit crossover map showing when and why each simulation paradigm breaks down;
An evaluation of simulation overhead introduced by basic error correction codes;
Quantified overheads associated with rudimentary error-correction

Together, these experiments will yield:

Multi-backend performance and accuracy profiles: Systematic comparison of ideal and noisy QFT and Shor subroutines across all simulators.
Crossover breakdown map: Identification of the qubit-count and depth regimes where each simulation paradigm fails or degrades sharply.
Error-correction overhead analysis: Measurement of the additional time and memory costs incurred by embedding basic quantum error-correction codes.
Noise-resilience quantification: Evaluation of how standard noise models impact simulator accuracy and resource usage.

An overview of this three-stage benchmarking workflow is illustrated in Figure 5.1.

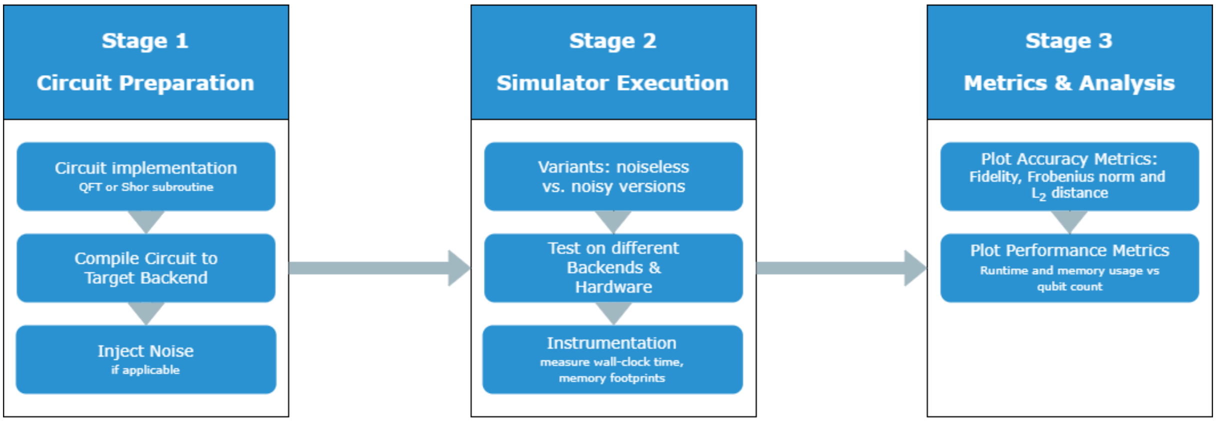
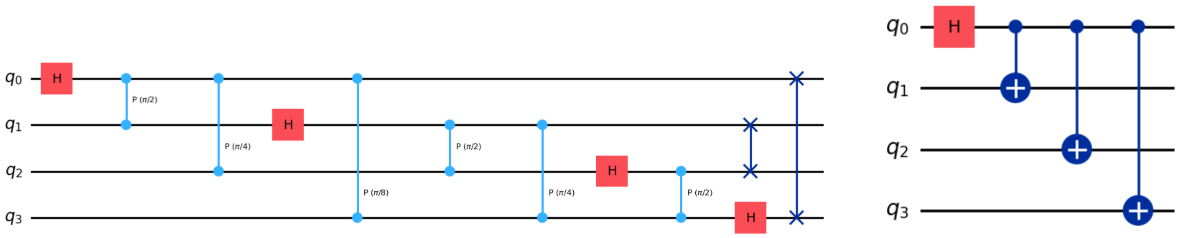

Stage 1
Circuit Preparation
Circuit implementation
QFT or Shor subroutine
Compile Circuit to Target Backend
Inject Noise
if applicable
Stage 2
Simulator Execution
Variants: noiseless vs. noisy versions
Test on different Backends & Hardware
Instrumentation
measure wall-clock time, memory footprints
Stage 3
Metrics & Analysis
Plot Accuracy Metrics:
Fidelity, Frobenius norm and L2 distance
Plot Performance Metrics
Runtime and memory usage vs qubit count

Figure 5.1: Overview of the benchmarking workflow. Stage 1 prepares the quantum circuits (e.g., QFT or Shor subroutines), compiles them to backend-specific formats, and optionally injects noise. Stage 2 executes both ideal and noisy variants across different simulators, recording run-time and memory usage. Stage 3 plots accuracy and performance metrics as a function of circuit size and backend.

5.3 Preliminary Results

This section presents an initial study of classical quantum circuit simulators, with a focus on evaluating performance under both ideal and noisy conditions. `Qiskit Aer` and `CUDA-Q` were tested on CPU and GPU backends, using the QFT as a representative algorithm.

5.3.1 Small-Scale QFT Simulation (4 Qubits)



(a) Qiskit circuit representation of the 4-qubit QFT circuit, including final SWAP gates to restore computational order.

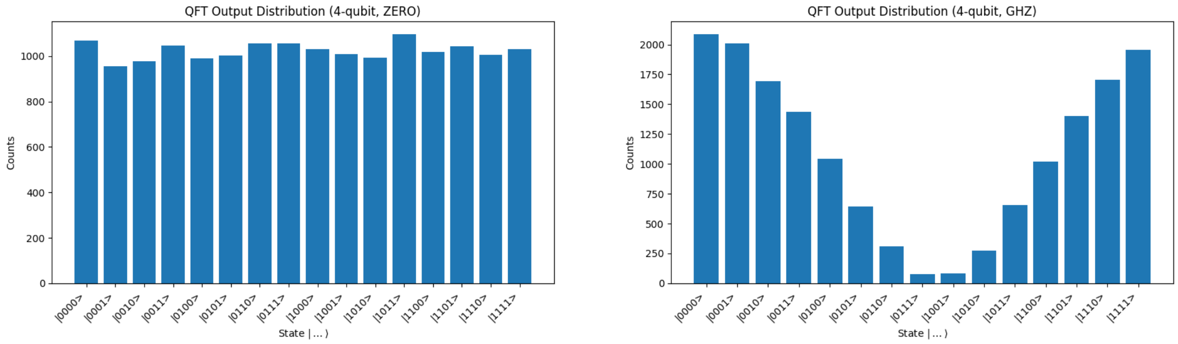
(b) Preparation circuit for the 4-qubit GHZ state used as the input to the QFT.

Figure 5.2: Quantum circuits used in the 4-qubit simulation study. (a) Full QFT circuit built using Qiskit's QFT class. (b) GHZ state initialisation circuit is applied before the QFT in entangled input experiments.

To validate the correctness of the QFT circuit implementation and illustrate its behaviour on simple quantum inputs, a 4-qubit simulation study was conducted using `Qiskit Aer` using `statevector` method. Two distinct initial states were considered: the product state $|0000\rangle$ and the 4-qubit entangled GHZ state. This provides a useful contrast between unentangled and entangled inputs and allows for direct comparison with the theoretical output distributions derived in Section 3.1.2.

The QFT circuit was generated using the `QFT` class from `qiskit.circuit.library`, and is shown in Figure 5.2(a). The GHZ input state was prepared using the subcircuit in Figure 5.2(b). Both circuits were executed with $131\,072$ shots.

The measured distributions are plotted in Figure 5.3. As expected, the $|0000\rangle$ inputs result in a nearly flat distribution over all 16 computational basis states, matching the uniform superposition predicted by theory. In contrast, the GHZ input leads to a non-uniform, symmetric distribution due to quantum interference from the superposition of $|0000\rangle$ and $|1111\rangle$. That reflects the phase structure from the exponential term $e^{2\pi i \cdot 15k/16}$ as derived in Equation (3.10).



(a) Uniform output from QFT applied to $|0000\rangle$.

(b) Interference pattern from QFT applied to $|\mathrm{GHZ}_4\rangle$.

Figure 5.3: Measured output distributions from the 4-qubit QFT circuit. The zero-initialised input yields a flat (uniform) distribution over all basis states, while the entangled GHZ input leads to constructive and destructive interference across outcomes.

The runtime and accuracy for each input are summarised in Table 5.1. As observed, the L_2 error is notably higher for the GHZ state. consistent with the increased entanglement complexity and its sensitivity to numerical and sampling noise.

Initial State	**Runtime (s)**	**L$_2$ Error**
$\|0000\rangle$	0.0071	0.0087
$\|\mathrm{GHZ}_4\rangle$	0.0080	0.1752

Table 5.1: QFT simulation performance on `AerSimulator` for different input states.

Overall, the experiment confirms the expected performance of the QFT implementation and provides a baseline for comparing larger-scale simulations under ideal and noisy conditions in the subsequent section.

5.3.2 No Noise Benchmark

In this section, the simulators are benchmarked under ideal (noiseless) conditions. The evaluation is carried out using two primary backends—`CUDA-Q` and `Qiskit Aer`—across various qubit counts

To select a shot count that balances statistical accuracy, measured in L_2 error, with runtime, the number of shots was varied from $4\ 096$ to $524\ 288$, and both runtime and L_2 error were measured for each simulator at its maximum qubit count.

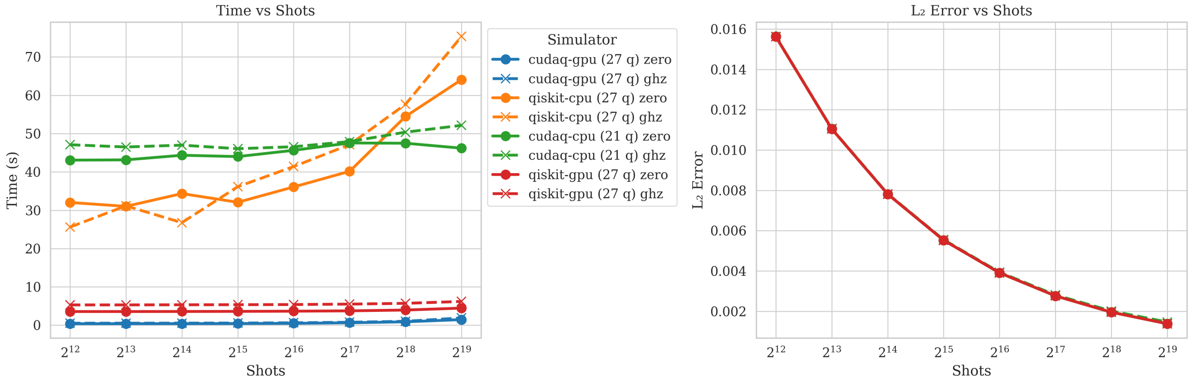


Figure 5.4: Effect of shot count on runtime (left) and L_2 error (right) at the maximum qubit count supported by each simulator.

For consistency across platforms and to ensure reliable accuracy without excessive computational cost, $131\ 072$ shots were selected as the default cutoff for all subsequent benchmarks.

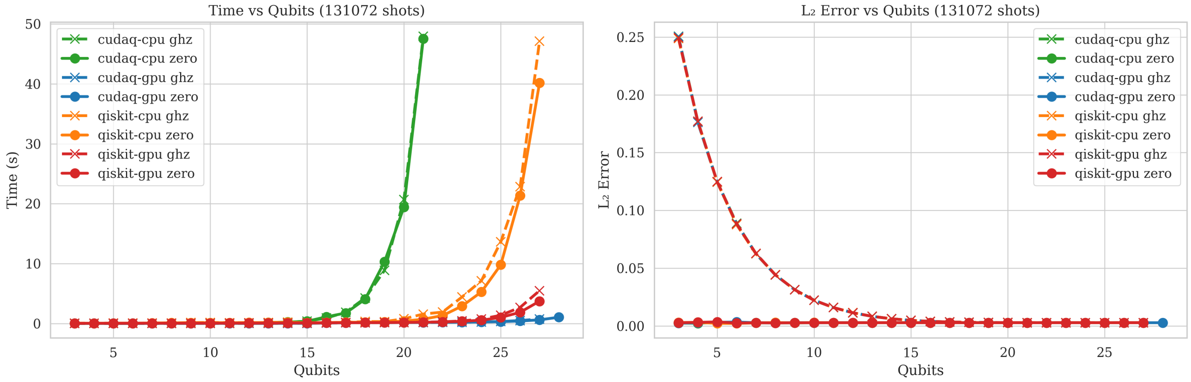


Figure 5.5: Runtime scaling across simulators (131072 shots). GPU back-ends sustain low runtime up to hardware limits.

Figure 5.5 illustrates how both runtime and L_2 error scale with the number of qubits. The GPU backends exhibit low execution times up to their respective memory limits. `CUDA-Q`'s GPU implementation stands out by maintaining near-constant runtime up to 28 qubits. This consistent performance can largely be attributed to the `cuStateVec` backend's ability to exploit GPU parallelism through techniques such as gate fusion, stream concurrency, and shot parallelism. In contrast, Qiskit Aer's GPU backend, while still efficient, begins to show moderate scaling beyond 25 qubits. This is likely due to memory bandwidth limitations, although the simulator remains competitive within the range tested.

For the CPU backend, the runtime grows exponentially with increasing qubit counts, which is expected due to the computational complexity of simulating large state-vectors on a CPU. However, Qiskit performs slightly better, as it starts scaling at around 21 qubits, compared to `CUDA-Q`, which begins to scale at around 15 qubits. This performance advantage of Qiskit can be attributed to its hybrid simulation strategy (Schrödinger–Feynman (SF), see Section 4.1.3).

One notable observation is that the runtime for both GHZ and zero initialisations remains almost identical. The difference, however, is evident in the L_2 error, where the GHZ state initially exhibits a higher error that then converges to zero. This indicates that both simulators are capable of achieving high accuracy.

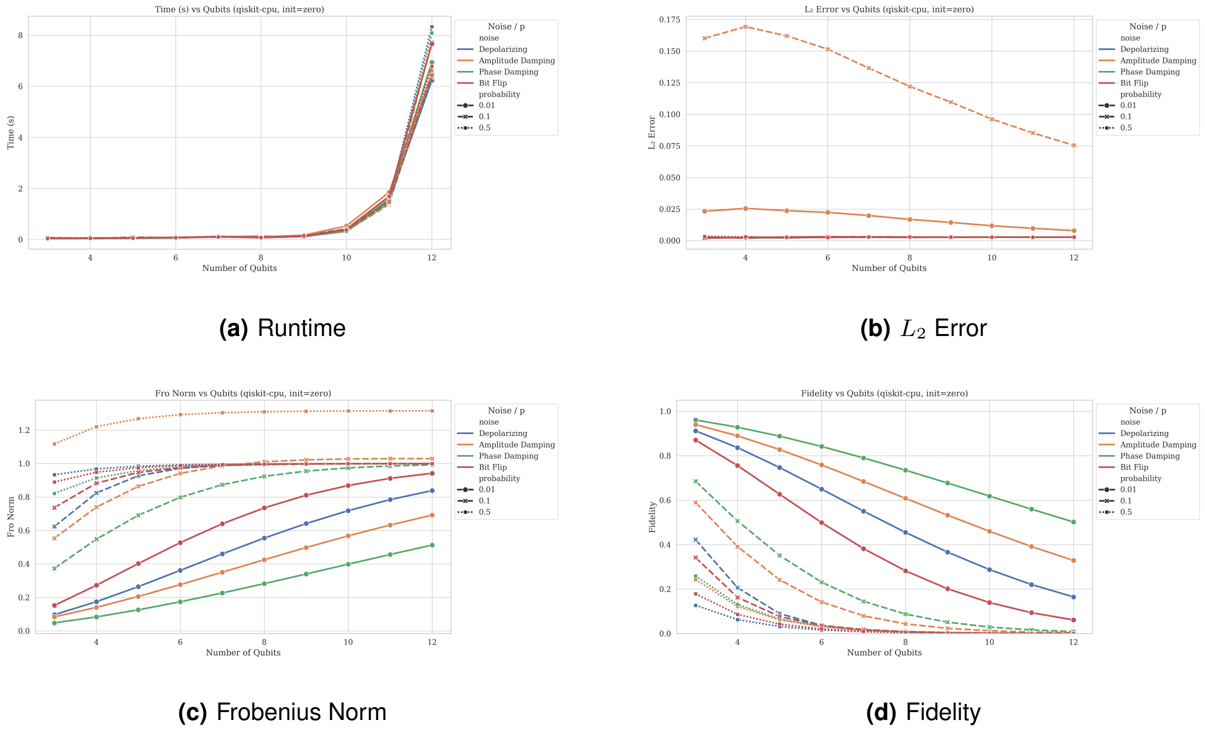
In conclusion, it can be observed that GPU acceleration plays a dominant role in optimizing the performance of quantum circuit simulators, particularly when using `CUDA-Q`.

5.3.3 Accuracy Under Noise

The performance of the `Qiskit Aer` CPU backend was evaluated under various noise models, including depolarising, amplitude damping, phase damping, and bit-flip. The evaluation tracked performance metrics—fidelity, Frobenius norm, L_2 error, and runtime—across QFT circuits with qubit counts ranging from 3 to 12, using $131\,072$ shots for each configuration.

The results indicate that fidelity degradation is most pronounced under depolarising and bit-flip noise, particularly at higher error probabilities, reflecting their disruption of both phase and amplitude coherence (Figure 5.6(d)). Phase damping, which only affects the phase, induces a more modest decline in fidelity, while amplitude damping causes a steady reduction as the system approaches the ground state. These trends are further supported by the Frobenius norm, where amplitude damping generates the largest deviation from unitary evolution (Figure 5.6(c)).

The L_2 error plot highlights that amplitude damping leads to the most significant error increase, especially at lower qubit counts, before stabilising as the system becomes more mixed. Other noise types contribute less to statistical divergence, particularly in Qiskit's implementation. Runtime scaling, as shown in Figure 5.6(a), remains mostly independent of noise model, with exponential growth occurring only beyond 10 qubits, suggesting that statevector expansion, rather than noise injection overhead, is the primary computational bottleneck.



(a) Runtime

(b) L_2 Error

(c) Frobenius Norm

(d) Fidelity

Figure 5.6: Performance of the `qiskit-cpu` simulator under four canonical noise models applied to a zero-initialised QFT circuit. Each colour corresponds to a noise type, while marker style encodes the error probability $p \in \{0.01, 0.1, 0.5\}$.

5.4 Work Plan and Timeline

The Grantt chart shown in Figure 5.7 outlines the main phases planned for the master's thesis, running from September 2025 to May 2026. The work is structured into five phases.

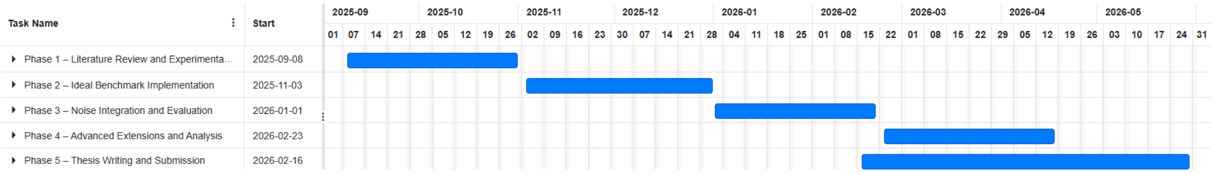
The first phase consists of a focused literature review aimed at surveying existing classical quantum simulators, available benchmarking suites, and defining the evaluation framework. By the end of Phase 1, a comprehensive checklist will be established detailing which circuits are to be tested on which simulators, under which conditions, with which parameters (such as shot counts, error rates, and qubit numbers), and what data should be captured. This preparation will ensure that the subsequent phases—benchmarking, noise analysis, advanced experiments, and final documentation—can proceed with a structured and consistent simulation pipeline across all platforms. The plan may be iteratively refined as the work progresses.

The second phase focuses on implementing the selected benchmark circuits under ideal conditions, compiling them for each simulator's native gate set and running noise-free simulations. The aim is to profile runtime, memory usage, and accuracy metrics while verifying correctness across platforms.

In the third phase, canonical noise models such as bit-flip, depolarising, amplitude damping, and phase damping will be injected into the circuits. Simulations will be re-run to assess the impact of decoherence on simulation accuracy using fidelity, Frobenius norm, and L_2 metrics.

Phase four explores optional but impactful extensions. These include evaluating basic QEC codes (e.g., the Steane code), running multi-GPU or distributed simulations, and—if time allows—testing AI-based circuit optimisations for noise robustness. This phase also consolidates performance vs accuracy results into the final crossover analysis.

The last phase is dedicated to writing, proofreading, and preparing the final submission.



Task Name	Start
Phase 1 – Literature Review and Experimenta…	2025-09-08
Phase 2 – Ideal Benchmark Implementation	2025-11-03
Phase 3 – Noise Integration and Evaluation	2026-01-01
Phase 4 – Advanced Extensions and Analysis	2026-02-23
Phase 5 – Thesis Writing and Submission	2026-02-16

Figure 5.7: Thesis work plan from September 2025 to May 2026.

$$|0000\rangle \xrightarrow{\text{QFT}} \frac{1}{4}\left(|0000\rangle + |0001\rangle + |0010\rangle + \ldots + |1111\rangle\right) = |++++\rangle$$

Bibliography

[1] IBM. (2023) IBM Quantum Roadmap for 2033. Accessed: 2025-05-23. [Online]. Available: https://www.ibm.com/quantum/blog/quantum-roadmap-2033

[2] ——. (2023) IBM Debuts Next-Generation Quantum Processor IBM Quantum System Two, Extends Roadmap to Advance Era of Quantum Utility. Accessed: 2025-05-23. [Online]. Available: https://newsroom.ibm.com/2023-12-04-IBM-Debuts-Next-Generation-Quantum-Processor-IBM-Quantum-System-Two,-Extends-Roadmap-to-Advance-Era-of-Quantum-Utility

[3] T. L. Scholten *et al.*, "Assessing the benefits and risks of quantum computers," arXiv preprint arXiv:2401.16317v2, 2024. [Online]. Available: https://arxiv.org/abs/2401.16317v2

[4] M. AbuGhanem, "Ibm quantum computers: evolution, performance, and future directions," *The Journal of Supercomputing*, vol. 81, p. 687, 2025. [Online]. Available: https://doi.org/10.1007/s11227-025-07047-7

[5] S. Efthymiou, S. Ramos-Calderer, C. Bravo-Prieto, A. Pérez-Salinas, D. García-Martín, A. Garcia-Saez, J. I. Latorre, and S. Carrazza, "Qibo: A framework for quantum simulation with hardware acceleration," *Quantum Science and Technology*, vol. 7, no. 1, p. 015018, December 2021. [Online]. Available: https://doi.org/10.1088/2058-9565/ac39f5

[6] T. Weber, K. Borras, K. Jansen, D. Krücker, and M. Riebisch, "Volumetric benchmarking of quantum computing noise models," *arXiv (Cornell University)*, Jan. 2023. [Online]. Available: https://arxiv.org/abs/2306.08427

[7] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79

[8] A. Holmes, A. Holmes, S. Johri, G. Guerreschi, J. Clarke, and A. Matsuura, "Impact of qubit connectivity on quantum algorithm performance," *Quantum Science and Technology*, vol. 5, 2018.

[9] R. Basili, W. Qian, S. Tang, A. Castellino, M. Eshaghian-Wilner, A. Khokhar, G. Luecke, and J. Vary, "Performance evaluations of noisy approximate quantum fourier arithmetic," *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 435–444, 2021.

[10] Smite-Meister, "Bloch sphere," https://commons.wikimedia.org/wiki/File:Bloch_sphere.svg, 2009, cC BY-SA 3.0.

[11] E. G. Rieffel and W. H. Polak, *Quantum Computing: A Gentle Introduction*, ser. Scientific and Engineering Computation. MIT Press, 2011.

[12] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano, and D. J. Wineland, "Demonstration of a fundamental quantum logic gate," *Phys. Rev. Lett.*, vol. 75, pp. 4714–4717, Dec 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.75.4714

[13] I. M. Georgescu, S. Ashhab, and F. Nori, "Quantum simulation," *Reviews of Modern Physics*, vol. 86, no. 1, p. 153–185, Mar. 2014. [Online]. Available: https://doi.org/10.1103/revmodphys.86.153

[14] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1994, pp. 124–134.

[15] H. de Boutray, H. Jaffali, F. Holweck, A. Giorgetti, and P.-A. Masson, "Quantum circuit representation of the quantum fourier transform for a n-qubit register," Figure 10 in "Mermin polynomials for non-locality and entanglement detection in Grover's algorithm and Quantum Fourier Transform", 2021, available from: https://www.researchgate.net/figure/Quantum-circuit-representation-of-the-Quantum-Fourier-Transform-for-a-n-qubit-register_fig10_349833228 [Accessed 7 Jun 2025].

[16] D. Gottesman, "The heisenberg representation of quantum computers," *arXiv (Cornell University)*, Jan. 1998. [Online]. Available: https://arxiv.org/abs/quant-ph/9807006

[17] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Physical Review A*, vol. 70, no. 5, Nov. 2004. [Online]. Available: https://doi.org/10.1103/physreva.70.052328

[18] S. Bravyi and D. Gosset, "Improved classical simulation of quantum circuits dominated by clifford gates," *Physical Review Letters*, vol. 116, no. 25, Jun. 2016. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.116.250501

[19] M. Van Den Nest, "Classical simulation of quantum computation, the gottesman-knill theorem, and slightly beyond," *arXiv (Cornell University)*, Jan. 2008. [Online]. Available: https://arxiv.org/abs/0811.0898

[20] ——, “Simulating quantum computers with probabilistic methods,” *arXiv (Cornell University)*, Jan. 2009. [Online]. Available: https://arxiv.org/abs/0911.1624

[21] L. Burgholzer, H. Bauer, and R. Wille, “Hybrid schrödinger–feynman simulation of quantum circuits with decision diagrams,” in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2021, pp. 119–130.

[22] A. Pasquale, A. Papaluca, R. M. S. Farias, M. Robbiati, E. Pedicillo, and S. Carrazza, “Beyond full statevector simulation with qibo,” 2024. [Online]. Available: https://arxiv.org/abs/2408.00384

[23] A. J. Gangapuram, A. Läuchli, and C. Hempel, “Benchmarking quantum computer simulation software packages: State vector simulators,” *SciPost Physics Core*, vol. 7, no. 4, Nov. 2024. [Online]. Available: http://dx.doi.org/10.21468/SciPostPhysCore.7.4.075

[24] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, no. 7779, pp. 505–510, 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1666-5

[25] S. Kim and I.-S. Suh, “Simulations of quantum approximate optimization algorithm on hpc-qc integrated systems,” *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02, pp. 464–465, 2024.

[26] J. Faj, I. Peng, J. Wahlgren, and S. Markidis, “Quantum computer simulations at warp speed: Assessing the impact of gpu acceleration: A case study with ibm qiskit aer, nvidia thrust & cuquantum,” *2023 IEEE 19th International Conference on e-Science (e-Science)*, pp. 1–10, 2023.

[27] Y. Bi, S. Xu, and Y. Ma, “Running qiskit on rocm platform,” *EPJ Web of Conferences*, 2024.

[28] H. Chaudhary, B. Mahato, L. Priyadarshi, N. Roshan, Utkarsh, and A. D. Patel, “A software simulator for noisy quantum circuits,” *International Journal of Modern Physics C*, 2019.

[29] H. Bayraktar, A. Charara, D. Clark, S. Cohen, T. B. Costa, Y.-L. L. Fang, Y. Gao, J. Guan, J. A. Gunnels, A. Haidar, A. Hehn, M. Hohnerbach, M. T. Jones, T. Lubowe, D. I. Lyakh, S. Morino, P. Springer, S. W. Stanwyck, I. Terentyev, S. Varadhan, J. Wong, and T. Yamaguchi, “cuquantum sdk: A high-performance library for accelerating quantum science,” *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, pp. 1050–1061, 2023.

[30] Y. Kim, A. McCaskey *et al.*, “Cuda-q: A platform for integrated quantum-classical computing,” *arXiv preprint arXiv:2308.08990*, 2023.

[31] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[32] A. H. Werner, D. Jaschke, P. Silvi, M. Kliesch, T. Calarco, J. Eisert, and S. Montangero, “Positive tensor network approach for simulating open quantum many-body systems,” *Phys. Rev. Lett.*, vol. 116, no. 23, p. 237201, 2016.

[33] A. Bassi and D.-A. Deckert, “Noise gates for decoherent quantum circuits,” *Physical Review A*, vol. 77, p. 032323, 2008.

A

Reproducibility and Code Availability

All of the benchmark scripts, circuit definitions, data-processing notebooks, and plotting code used in this work are hosted at:

https://github.com/madalenarb/PIC2_QuantumComputing/