Quantum Technology final report : Quantum algorithms

Nahomé Vesvard

December 2025

Abstract

This report summarizes a quantum algorithms laboratory carried out with Qiskit on IBM Quantum hardware and simulators. Following the structure of the provided Python notebook, we revisit each step in greater detail and compare ideal, noisy, and hardware executions for a Quantum Random Number Generator and a phase-decoding experiment based on the inverse Quantum Fourier Transform. The results highlight how noise, limited connectivity, and two-qubit gate affect circuit depth and success probability, and why performance degrades as circuits scale. We conclude by discussing the main error mechanisms observed in practice and the broader challenges they pose for running larger quantum algorithms on current devices.

1 Introduction

Quantum computing studies how information processing changes when computation is performed using quantum-mechanical systems. The basic unit is the qubit, a two-level quantum system whose state can be a superposition of computational basis states. When multiple qubits interact, their joint state can exhibit entanglement, and quantum interference allows probability amplitudes to combine constructively or destructively. Quantum algorithms exploit these effects by engineering interference so that the correct solution is obtained with higher probability than other outcomes. Since measurement is intrinsically probabilistic, the same quantum circuit is typically executed many times ("shots") and the results are collected as a histogram; the most frequent outcomes provide an estimate of the circuit's output distribution and (ideally) reveal the expected answer.

In practice, today's quantum computers operate in a hybrid workflow: a classical computer is used to construct and transpile a quantum circuit into the device's native gate set and connectivity constraints, execute it on a quantum processing unit (QPU), and then post-process the measurement results. In this lab we use Qiskit, IBM's Python framework, to design circuits, run simulations, and execute programs on IBM Quantum hardware while inspecting backend properties such as qubit connectivity and calibration data.

We investigate two representative algorithmic building blocks: a Quantum Random Number Generator (QRNG) based on measurement of superposition states, and the Quantum Fourier Transform (QFT) combined with a single phase oracle on 6–8 qubits. Finally, we study how noise sources (e.g., readout error, two-qubit gate errors, and decoherence) affect experimental outcomes by comparing ideal simulation, noisy models, and real-device results.

2 Measurements

2.1 Software and environment

Prior to running our quantum algorithms, we set up the Qiskit environment. We use three layers of the Qiskit stack:

1. qiskit (v2.2.1): used to write the code, build quantum circuits, and transpile them.
2. qiskit-aer (v0.17.2): simulators used to run circuits without real hardware.
3. qiskit-ibm-runtime (v0.42.0): package used to connect Qiskit to IBM Quantum backends and execute circuits on real devices.

We therefore use two types of backends throughout this work: simulated backends (local Aer simulators) and real hardware backends (IBM Quantum devices accessed via the Runtime service). This allows direct comparison between ideal/simulated results and outcomes measured on noisy hardware.

2.2 Backend inspection

After authenticating with the IBM Quantum Platform, we can query the backends available to us. Among the real quantum devices, only two backends are accessible in our configuration: ibm torino (133 qubits) and ibm brisbane (127 qubits). Both provide the same set of nine basis operations (native instructions).

In addition to hardware devices, several simulated backends are available, with supported qubit counts ranging from 1 up to 133.

For the purposes of this experiment, only a limited number of qubits is required (up to 8 qubits for the

QFT circuit). Therefore, either hardware backend is suitable. We selected the backend with the shortest current queue time, which at the time of execution was ibm brisbane. Consequently, our hardware runs were performed on a 127-qubit device with nine native operations.

We now examine the backend structure in more detail (in particular its qubit connectivity and native gate set).

The 9 elementary operations and their actions are :

- X – NOT gate : flips the computational basis states $|0\rangle \leftrightarrow |1\rangle$
- $Reset$: re-initializes a qubit to $|0\rangle$
- $S_x - \sqrt{X}$ gate : x- axis rotation by $\pi/2$
- $Measure$: Measures a qubit in the computational (Z) basis and writes the result to a classical bit (collapses the state to $|0\rangle$ or $|1\rangle$
- ECR : native 2-qubit entangling gate used to implement the CNOT gate by combining it to single-qubit gates
- $delay$: Explicitly wait for a duration τ
- id – identity : “Do nothing” unitary, state ideally unchanged
- $ifelse$: works as the classical control-flow operation
- R_z – Rotation around the Z axis by angle θ : it doesn’t change measurement probabilities in the Z basis directly, but it changes relative phase, which affects later interference.

Furthermore, we can check the connectivity of our qubits by looking at the coupling map.

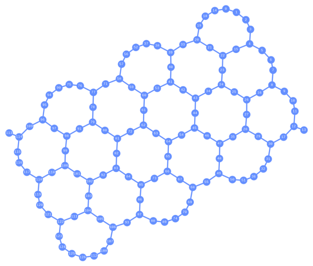

Figure 1: IBM Brisbane coupling map

Figure 1 represents the real wiring on the physical chip. As we can see our qubits are displayed along a honeycomb structure, which means not all qubits are directly linked to each other.

This honeycomb (heavy-hex) connectivity matters especially for multi-qubit operations. Two-qubit gates can only be applied directly between connected (adjacent) qubits in the coupling map. Therefore, if an algorithm requires entangling qubits that are not neighbors, the transpiler must insert SWAP operations to move the quantum states along available connections until the desired pair becomes adjacent.

We will begin our study by building a simple logical 3-qubit circuit (Cf figure 2).

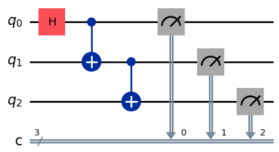

Figure 2: Introductory 3-qubit circuit : prepares one qubit in a superposition and applies two-qubit interactions to correlate it with the other qubits

This simple case will enable us to get familiar with native gates and coupling.

2.3 circuit 1: QRNG

Now that the quantum computing system has been characterized (available backends, native operations, and connectivity), we can move on to the first experiment: a Quantum Random Number Generator (QRNG). As its name suggests, a QRNG generates random numbers by exploiting the inherent randomness of quantum measurement outcomes.

To obtain uniformly random bits, each qubit is prepared in an equal superposition of $|0\rangle$ and $|1\rangle$ by applying a Hadamard gate:

$$|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$$

So that measuring $|+\rangle$ in the computational basis ideally yields 0 and 1 with probability $\frac{1}{2}$.

However, this gate is not part of the 9 native operations of our harware backend. During transpilation it is therefore decomposed into native single-qubit gates, for example:

$$H \;\equiv\; R_z\!\left(\tfrac{\pi}{2}\right)\, S_x\, R_z\!\left(\tfrac{\pi}{2}\right) \text{ (up to a global phase)}$$

In Qiskit, the Hadamard is available directly as the circuit instruction ’h’, which the transpiler later maps to the backend basis.

We implement the QRNG on $n = 3$ qubits, producing 3-bit strings upon measurement. Ideally, the output distribution is uniform over the $2^3 = 8$ possible outcomes, from $000 \equiv 0$ to $111 \equiv 7$. The circuit is executed on both a simulated backend and a real hardware backend for comparison; the resulting distributions are analyzed in the next section.

2.4 circuit 2 : QFT oracle

The Quantum Fourier Transform (QFT) is a unitary change of basis, analogous to the discrete Fourier transform, and is mainly used as a “decoder” in quantum algorithms: it can convert information stored in the relative phases of a quantum state into amplitude differences, so that a standard measurement in the computational basis returns the desired bitstring with high probability.

This is important because intermediate results in quantum circuits are often phase-encoded: applying phase factors to components of a superposition does not directly change measurement probabilities, but it does change how the amplitudes will interfere after subsequent gates.

Many algorithms deliberately exploit this idea, where a controlled operation imprints phase information onto a register without changing its computational-basis values. The problem-specific part of such circuits is commonly modeled as an oracle, which is a black-box unitary that encodes a function or property of interest. For instance, taking a boolean function f such as:

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

One can define:

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

and the equivalent phase-oracle form:

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

or more generally:

$$U_\phi |x\rangle = e^{i\phi(x)} |x\rangle$$

In this setting, applying the (inverse) QFT after the oracle transforms the phase pattern created by the oracle into a measurable output distribution, making the encoded information accessible through repeated measurements.

The QFT on n qubits has a very regular layered structure. It is built by processing the qubits one by one: on each qubit we first apply a Hadamard gate to create a superposition, and then we apply a sequence of controlled phase rotations from the remaining qubits. These controlled rotations add relative phases whose angles decrease by powers of two, which is the quantum analogue of the complex phase factors that appear in the classical discrete Fourier transform. Repeating this pattern across all qubits produces the full QFT unitary, but the output qubits end up in reversed significance order (least significant becomes most significant). For that reason, the circuit is usually finished with a bit-reversal step, implemented as a sequence of SWAP gates, to restore the conventional bit order.

In our experiment, we tested phase decoding by applying an inverse QFT after a phase oracle that imprints a known single-tone phase pattern onto registers of $n = 6$ and $n = 8$ qubits. Schematically, our circuit is therefore doing :

$$H^{\otimes n} \to \text{Oracle}(k) \to \text{QFT}^{-1}$$

Before running our code on these backends, we performed a sanity check using the classical–quantum analogy with the Fourier transform. Classically, we generated the reference signal $x[t] = e^{\frac{2i\pi kt}{N}}$ and computed its FFT, which should exhibit a dominant peak at frequency bin k. In the quantum case, we built the ideal phase-decoding circuit.
The circuits were executed both on our selected real backend and on a noisy Aer simulator configured with depolarizing noise on gates and a readout-error model for comparison.

3 Analysis

3.1 Backend/topology implications

As seen in the Measurement section, the placement of our qubits is crucial because it directly impacts the circuit depth. The farther apart two qubits are in the coupling map, the more SWAP operations are required to bring them next to each other before a two-qubit gate can be applied.

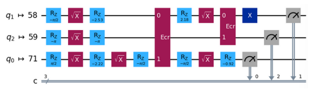

Figure 3: Logical circuit Mapped

In Figure 3 we show the transpiled version of our introductory circuit, which has a depth of 11 and contains 22 basis operations. Because physical qubits 71–58–59 form a contiguous chain in the coupling map, this layout is well matched to the circuit's interaction pattern and does not require additional SWAP operations for routing. More generally, the choice of physical qubits (initial layout) is an important practical factor: depending on the device connectivity, the same logical circuit can transpile to different depths and different numbers of two-qubit gates.

One major consequence of circuit depth is the raise of the overall error rate which causes the probability of measuring the correct output to fall. This effect is particularly relevant for QFT circuits, which require many controlled two-qubit interactions and therefore can suffer from large routing overhead on limited-connectivity hardware.

3.2 QRNG results

After running our QRNG on both real and Aer simulated backend, we analyze our results through statistical computations.

Focusing first on the real backend, we begin by testing the randomness of our circuit. For each qubit we collect a bitstring of length S=4096 from repeated executions (shots) and apply two simple statistical checks. The monobit test reports $p1$ the fraction of 1s (ideal $p1 = 0.5$) along with a z-score z measuring any bias, while the runs test counts the number of consecutive blocks of identical bits ("runs") and returns a z-score zr that highlights possible short-range correlations.

	z	zr
q0	-0.53	0.66
q1	1.00	2.08
q2	0.09	1.28

Table 1: z scores of monobit and run test for q0, q1 and q2

The monobit z-scores are small for all three qubits, indicating no significant bias in the proportion of 0s vs 1s. The runs test is also consistent with randomness for $q0$ qnd $q2$ while $q1$ shows the largest deviation ($zr = 2.08$, suggesting a mild departure from ideal independence (possible weak correlations or readout effects).

Secondly we displayed the QRNG results under a histogram with bins for each possible bitstring outcome for both backends.

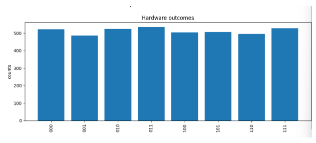

Figure 4: Hardware QRNG histogram results

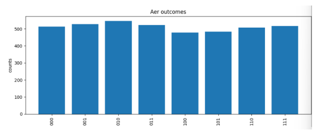

Figure 5: Simulator QRNG histogram results

We expect to get about $\frac{4096}{8} = 512$ counts for each bin. As visible on figure 4 and 5, the distribution is pretty much uniform over the 8 outcomes for both simulated and hardware backends. The QRNG behaves approximately ideally: the deviations from perfectly flat are small enough that they can be explained by ordinary sampling randomness, with minor additional noise from the device.

To analyze the QRNG beyond simple histograms, we next compute temporal correlations (correlations between successive shots of the same qubit) and spatial correlations (correlations between different qubits within the same shot) for hardware results. This is relevant because an output can look nearly uniform while still exhibiting memory effects or interqubit crosstalk; correlation measures help reveal these non-idealities on hardware. For conciseness, we show a representative example in Figure 6; the remaining plots are omitted since the pattern for q0, q1 and q2 is similar.

Considering we did $S = 4096$ shots, the expected statistical fluctuation of the sample autocorrelation

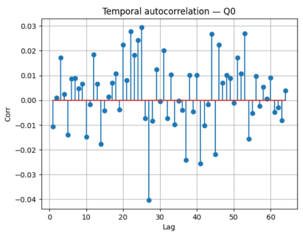

Figure 6: Temporal correlation of q0 in the QRNG circuit

under the hypothesis of independent bits is approximately $\sigma \approx \frac{1}{\sqrt{N}} = \frac{1}{\sqrt{4096}} = \frac{1}{64} \approx 0.016$. A commonly used 95% confidence band is therefore given by $\pm 1.96\,\sigma \approx \pm 0.031$.

Since the observed autocorrelation values remain within $[-0.04; 0.04]$, they are of the same order as the expected finite-sampling fluctuations, with no clear evidence of strong temporal correlations. Small excursions around zero can occur purely due to finite statistics, especially when many lags are evaluated simultaneously.

We can use the same argument regarding spatial correlation, using the values displayed in the spatial correlation matrix below :

$$\begin{pmatrix} 1 & -0.009 & 0.015 \\ -0.009 & 1 & 0.019 \\ 0.015 & 0.019 & 1 \end{pmatrix}$$

Given the small magnitude of the off-diagonal terms of the correlation matrix (near zero), there is no apparent spatial-correlation structure.

However, correlation effects can still arise in practice due to noise sources such as readout errors, crosstalk, and drift; in the Noise subsection we discuss how these mechanisms can generate temporal and spatial correlations and how they impact circuit performance.

3.3 QFT and phase oracle results

As expected, the probability of measuring $|k\rangle$ in the ideal (noise-free) case is close to 1, which validates our implementation of the phase oracle and inverse QFT.

Our next focus is how the QFT scales with the register size. Table 2 summarizes the main resource metrics for two register sizes, after transpiling the QFT pipeline on the real backend.

n	Depth	Two-qubit gates
6	118	46
8	201	93

Table 2: Scaling of inverse QFT circuit according to number of qubit n

From Table 2, increasing the register size from n = 6 to n = 8 leads to a strong increase in transpiled resource cost: the depth grows from 118 to 201 (about 70%), and the number of two-qubit gates rises from 46 to 93 (roughly a factor of 2). This scaling is expected because the inverse QFT is built from layers of Hadamard gates and, more importantly, many controlled-rotation operations whose count grows rapidly with n (approximately quadratically). In addition, the compiled cost depends on the backend's native gate set: while single-qubit operations are relatively cheap, controlled rotations must be decomposed into the available native two-qubit entangling gate (here the backend uses ECR), plus additional single-qubit rotations. As a result, the specific choice and implementation quality of the native two-qubit gate strongly affects how efficiently the QFT can be transpiled, and helps explain why depth and two-qubit gate counts increase so quickly as n grows.

We now examine the results obtained with the noisy Aer simulator. The noise model includes depolarizing noise applied to all one-qubit gates with probability $p1 = 0.001$, depolarizing noise applied to all two-qubit gates with probability $p2 = 0.01$, and a readout error with probability $pr = 0.02$. In this study we choose the oracle parameter $k = 37$; in the ideal (noise-free) case, the measured output should be strongly peaked on the bitstring corresponding to k, i.e. $P(measure = 37) \approx 1$.

However, as we could expect with the noise implementation, we get:

$$P(measure = 37) \approx 0.76 \text{ for } n = 6$$
$$P(measure = 37) \approx 0.67 \text{ for } n = 8$$

Thus, increasing the register size reduces the probability of measuring the target state $|k\rangle$. This is consistent with the scaling shown in Table 2: going from $n = 6$ to $n = 8$ increases the circuit depth ($118 \to 201$) and roughly doubles the number of two-qubit gates ($46 \to 93$), which increases the accumulated effect of noise. As a simple heuristic, if each operation fails independently with probability p, then the overall success probability scales roughly as

$$P_{success} \approx (1 - p)^{N_{ops}}$$

Finally, for our last experiment we run our QFT pipeline on the real hardware, and this time the probability of measuring $k = 37$ drops to roughly 0.01.

3.4 Noise-focused comparison

In the ideal (noise-free) simulation, the phase-decoding pipeline is correct by construction: for a single-tone oracle parameterized by k, the inverse QFT concentrates the final state on $|k\rangle$, so $P(measure = k) \approx 1$ up to finite-shot fluctuations. When moving to the noisy Aer simulator, this success probability drops because we explicitly inject errors at each operation (depolarizing noise on one- and two-qubit gates, plus readout error). This matches our earlier scaling results: increasing n increases the transpiled depth and, in particular, the number of two-qubit gates, so the accumulated noise grows and the output distribution spreads away from the ideal peak at $|k\rangle$.

On real hardware the performance is typically worse (and more variable) than in the toy noisy simulation, because additional effects are present and are not fully captured by simple noise channels. The dominant error mechanisms include:

- Decoherence (finite energy-relaxation time T_1 i.e. how fast $|1\rangle$ decays to $|0\rangle$ and dephasing time T_2 i.e. how fast a superposition loses its relative phase) - which thus introduces relaxation and dephasing during the circuit runtime.
- Gate imperfections - especially on two-qubit gates (e.g., over/under-rotations and control errors).
- Readout errors that flip measured bits or bias outcomes.
- Routing and long-range interactions: QFT circuits require many controlled two-qubit interactions between qubits that are far apart in the logical register, and limited connectivity can force additional routing that increases depth and two-qubit gate count.

Beyond these local effects, real hardware can exhibit additional consequences such as leakage (population leaving the computational subspace), crosstalk and residual couplings (ZZ interactions between qubits or interference between simultaneous operations), and calibration drift over time. These correlated and time-dependent mechanisms become more impactful for deeper circuits like the QFT, and help explain the remaining gap between the noisy-simulator baseline and real-device results, even though our QRNG tests showed near-zero temporal and spatial correlations over 4096 shots.
To reduce these effects, circuit-level mitigations can be applied; for example dynamical decoupling on idle qubits to suppress dephasing during long idle periods [2], and readout error mitigation (assignment-matrix calibration) [3] to reduce measurement bias; both are expected to increase $P(measure = k)$ and bring the observed distributions closer to the ideal case.

4 Conclusions

Overall, the QRNG behaved as expected on hardware: the 3-bit output distribution was close to uniform with only small deviations consistent with finite-shot fluctuations and minor device imperfections. The QFT and phase-oracle pipeline was validated in the ideal case (output peaked at $|k\rangle$), but its success probability decreased for $n = 8$ compared to $n = 6$ as circuit depth and, especially, two-qubit gate count increased. This indicates that two-qubit gate errors (and the extra routing they induce), together with decoherence over longer runtimes, are the main factors limiting performance, while readout error mainly adds smaller biases.

A practical takeaway is that backend connectivity and two-qubit gate quality become increasingly important as circuits deepen; future improvements include reducing circuit depth (using for instance approximate QFT or layout optimization) and applying mitigation such as readout correction and dynamical decoupling.

More broadly, a central challenge for today's Noisy Intermediate-Scale Quantum (NISQ) devices is the trade-off between scale and reliability [4]: adding more qubits increases the size of problems we can represent, but it often also increases depth, routing overhead, and accumulated noise, so scaling qubit count alone does not automatically lead to more reliable computations. Similarly, higher connectivity could reduce SWAP insertion and routing overhead, but achieving denser coupling in hardware is challenging due to crosstalk and calibration constraints.

Finally, progress beyond the NISQ regime ultimately requires fault-tolerant operation [5], making quantum error correction one of the key long-term challenges for scalable quantum computing.

5 References

References

[1] Circuit Library - IBM Quantum Platform

[2] Pad Dynamical Decoupling - IBM Quantum Platform

[3] Error mitigation and suppression techniques - IBM Quantum Platform

[4] Quantum computing in the NISQ era, J. Preskill (2018)

[5] Making fault-tolerant quantum computers a reality - McKinsey & Company H. Soller, M. Gschwendtner, V. Kermans (2025)