Supplementary material for *Implementing Quantum Polar Codes in a Superconducting Processor: From State Preparation to Decoding*

Handy Kurniawan[1], Valentin Savin[2], Carmen G. Almudéver[3], and Francisco García Herrero[1]

[1]Universidad Complutense de Madrid, Madrid, 28040, Spain
[2]Université Grenobles Alpes, CEA-Léti, F-38054, Grenoble, France
[3]Universitat Politècnica de València, Valencia, Spain

Contents

1 Quantum Error Correction

Quantum error correction (QEC) is a fundamental technique for protecting quantum information from noise and other errors that inevitably arise during quantum operations and storage. Unlike classical error correction, QEC must address the unique challenges of the quantum realm, including the no-cloning theorem [1] and the inability to directly measure qubits without collapsing their quantum states [2]. To overcome these challenges, quantum error-correcting codes (QECCs) encode quantum information into redundant qubits, enabling error detection and correction without directly disturbing the encoded quantum state [3].

This section provides an overview of quantum errors and the basic definitions involved in QECC for the potential readers of the paper "Implementing Quantum Polar Codes in a Superconducting Processor: From State Preparation to Decoding" who are unfamiliar with the area.

1.1 Main sources of error

Quantum noise can affect stored quantum information and may also cause faulty behavior in quantum gates, state preparation, or measurements [3]. Accordingly, the main sources of errors that are considered on a simple quantum noise model are:

1. **Qubit errors**: arising from decoherence and other forms of quantum noise caused by interactions between a qubit and its surrounding environment. These errors often manifest as a transition to a mixed or entangled state with the environment, leading to the degradation of the information stored in the qubit.

2. **Gate errors**: occur when a quantum gate fails to apply the intended operation on the qubit(s), causing an incorrect transformation of the quantum state.

3. **Measurement errors**: occur when the classical outcome of a quantum measurement does not accurately reflect the quantum state being measured, leading to incorrect results.

More sophisticated error models may also account for various forms of error correlations arising from a shared environment. Additionally, crosstalk errors can occur when a qubit unintentionally interacts with a neighboring qubit, causing unintended state changes due to physical proximity or shared control signals.

1.2 Basic QECC definitions

A QECC is denoted as $[[n, k, d]]$, where n is the number of physical data qubits (or code length), k is the number of logical qubits (or code dimension), and d is the minimum distance of the code, which determines its error detection and correction capability. In this context:

- A physical qubit is a qubit implemented on a real quantum processor, that is, a physical device that behaves as a two-state quantum system, and which is subject to quantum noise.

- A logical qubit is an encoded representation of quantum information, constructed from multiple physical qubits, referred to as physical data qubits.

Errors occurring in the encoded state are identified using syndrome measurements, which extract information about physical errors without collapsing the quantum state. This requires the use of auxiliary physical qubits, known as ancilla qubits, which interact with the physical data qubits in a specific, code-dependent manner, and are subsequently measured. The extracted classical bits (syndrome) are processed by a classical decoding algorithm, which aims to identify the error that has happened, allowing the system to be restored to the correct logical state. To ensure effective protection against errors, the QECC must guarantee that the probability of logical errors is lower than that of physical errors. A key ingredient for such a guarantee is the fault tolerance of the syndrome measurement circuit. Precisely, the goal is to prevent physical errors from propagating uncontrollably during syndrome measurement, as this could generate more errors than the code can correct. Assuming fault tolerant syndrome measurement, minimum distance bounded decoding, and a sufficiently small physical error rate (PER) p, the logical error rate (LER) is proportional to $p^{\lfloor (d+1)/2 \rfloor}$, ensuring exponential error suppression as the code distance d increases.

Most QECCs rely on the stabilizer formalism [4], in which the quantum state is encoded in the $+1$ eigenspace of a group of commuting Pauli operators, referred to as stabilizer group. The key principle is that the operators in the stabilizer group do not change the encoded state, but they can reveal errors when measured. Thus, syndrome measurement can be performed by directly measuring a set of generators of the stabilizer group. However, this procedure is not inherently fault-tolerant, as it can propagate errors. Yet, if the stabilizer group has a set of low-weight generators, fault-tolerance can be achieved by performing repeated syndrome measurements, where errors are detected by the

difference between syndromes measured in consecutive rounds. This approach is used to perform fault-tolerant error correction with quantum low-density parity-check (LDPC) codes, including topological constructions such as such as the Kitaev's toric code [5] or Bombin's color codes [6]. The planar version of these latter codes, the well-celebrated surface [7] and triangular color codes [8], are of particular interest because their qubit layout is reduced to a two-dimensional grid, making them easily mappable to physical systems.

For Calderbank-Shor-Steane codes (CSS) codes, a subclass of stabilizer codes with stabilizer group generated by only type-X and type-Z Pauli operators, syndrome measurement is closely related to the notion of logical state preparation. The latter notion refers to the ability to prepare an eigenstate of a logical Pauli operator, such as logical $|0\rangle_L$ or $|+\rangle_L$. Such a logical state can be prepared by initializing all the physical data qubits in either $|0\rangle$ or $|+\rangle$, and then measuring either the type-X or the type-Z generators, respectively, of the stabilizer group. If these measurements can be performed in a fault-tolerant manner (*e.g.*, for the quantum LDPC codes mentioned above, by using ancilla qubits and repeating the measurements multiple times), then the preparation procedure is also fault-tolerant. Otherwise, it is not. Conversely, if we are given a logical state, say $|0\rangle_L$, prepared on an auxiliary quantum system, we can compute the Z-error syndrome by using the Steane technique [9,10]. Precisely, Z-errors are copied from the original to the auxiliary system by using transverse CNOT gates, the physical data qubits of the auxiliary system are then measured in X-basis, and the Z-error syndrome is computed classically. Note that the original system does not use any ancilla qubits in this case, as we do not directly measure the syndrome on it. Instead, an auxiliary system, which must be prepared in a logical state, is used. If we have a fault-tolerant procedure to prepare logical states, we can then use the Steane technique to measure the syndrome in a fault-tolerant manner. This approach is practical for codes with high-weight generators, where directly measuring these generators would not be fault-tolerant. In particular, this is the approach used for quantum polar codes, which are discussed in the next section.

1.3 Quantum polar codes

Polar codes are a family of error correcting codes of significant interest in both classical and quantum error correction. First introduced first by Arikan in 2009 for classical channels [11], and later generalized to the quantum case [12–15], polar codes arguably represent one of the most important advances in coding theory of the last fifteen years. In the classical setting, they were the first family of codes to achieve the symmetric capacity of any binary-input discrete memoryless channel, with log linear encoding and decoding complexity. In the quantum setting, they are known to achieve the symmetric coherent information of any quantum channel, while inheriting the low complexity decoding algorithm of their classical counterparts.

In this work, we focus on CSS quantum polar codes [12], which exploit classical polarization in both the X or Z bases. For CSS quantum polar codes encoding one logical qubit, referred to as Q_1 codes, a fault-tolerant logical state preparation procedure has been proposed in [16]. Such a procedure is essential for fault-tolerant error correction using the Steane method, as explained in Section 1.2. Moreover, a factory-based extension of this preparation procedure has been proposed in [17], where it has been shown that a Q_1 code may provide a logical error rate improvement of about three orders of magnitude compared to a surface code with similar length and minimum distance.

The logical state preparation procedure proposed in [16] follows a measurement-based protocol, where n rounds of two-qubit Pauli measurements are performed to prepare a code with $N = 2^n$ physical data qubits (note the difference in notation with respect to Section 1.2). Importantly, part of the measured Pauli operators do not belong to the stabilizer group. Thus, the protocol involves the measurement of anti-commuting two-qubit Pauli operators, which progressively build the stabilizer group of the code. As a consequence, these measurements cannot be repeated to ensure fault tolerance. Instead, the fault tolerance is guaranteed by an error detection mechanism, where, if an error is detected, the preparation procedure is discarded and restarted from the beginning. Due to error detection, the preparation is probabilistic, and its success rate is referred to as the preparation rate. A high preparation rate is desirable to minimize the number of discarded states, optimizing time or space resources, depending on whether logical states are being prepared sequentially or in parallel.

Implementing the state preparation protocol on a device with only near-neighbor connectivity requires introducing additional SWAP gates to route qubits between non-adjacent locations, ensuring that all necessary qubit interactions can be performed. We note that when the error detection mecha-

nism is present, such SWAP gates do not affect the fault-tolerance property of the measurement-based circuit. This follows from (the technical details in the proof of) [16, Theorem 3]. However, SWAP gates introduce additional noise, which may degrade the preparation rate, thereby motivating the use of noise-aware compilation techniques. Beyond the purely QEC aspects, considering the preparation rate as a key performance metric to assess how robust the compiled circuit is to noise can also serve as a broader demonstration of the significant benefits that noise-aware compilation techniques may bring to the implementation of quantum circuits on NISQ devices.

We conclude this section with a discussion on the weight of the Pauli operators in the stabilizer group of a Q_1 code, showing that, as stated in the paper, these codes do not admit a set of small-weight generators. For a given Q_1 code and a set of stabilizer generators G, we define $\mathrm{wt}(G) := \max\{\mathrm{wt}(g) \mid g \in G\}$, where $\mathrm{wt}(g)$ is the number of qubits on which g acts non-trivially. We further define $\mathrm{wt}(Q_1) := \min_G \mathrm{wt}(G)$, where the minimum is taken over all the sets of generators of the stabilizer group. For additional details, we shall also use subscripts X or Z to refer to the specific type of stabilizer generators, *e.g.*, $\mathrm{wt}_X(Q_1)$ or $\mathrm{wt}_Z(Q_1)$. The $\mathrm{wt}(Q_1)$ value depends on the specific Q_1 code, that is, the code length N and the position $i \in \{0, \ldots, N-1\}$ encoding the logical information (see [16] for details). If the code length N is not too large, the $\mathrm{wt}(Q_1)$ can be computed numerically, or at least a lower bound can be determined. Considering the Q_1 codes from [16]:

- For the $Q_1(N = 16,\ i = 6)$ code, we have $\mathrm{wt}_X(Q_1) = 8$ and $\mathrm{wt}_Z(Q_1) = 4$. For the X-type generators, the set of generators G_X minimizing $\mathrm{wt}(G_X)$ contains eight generators of weight 2 and one generator of weight 8. The eight generators of weight 2 generate all the X-type stabilizer operators of weight ≤ 6, hence, any set of generators must contain at least a generator of weight greater than or equal to 8.
- For the $Q_1(N = 16, i = 3)$ code, corresponding to a Shor code, we have $\mathrm{wt}_X(Q_1) = 2$ and $\mathrm{wt}_Z(Q_1) = 8$. This code does not have Z-type generators of weight less than 8.
- For the $Q_1(N = 64, i = 22)$, it can be verified that both $\mathrm{wt}_X(Q_1) \geq 8$ and $\mathrm{wt}_Z(Q_1) \geq 8$.

2 Noise-Aware Compilation: Implementation Details

In this section, we provide details on the noise-aware (NA) compilation techniques implemented in this work, highlighting their role in mitigating the impact of noise in quantum circuits.

2.1 Noise models

In quantum computing, a noise model characterizes the imperfections and errors that occur during the execution of quantum operations on real hardware. These errors stem from various sources, including gate inaccuracies, decoherence, measurement, and crosstalk errors (see Section 1.1). Noise models are essential tools for simulating the behavior of quantum circuits under realistic conditions. They enable us to evaluate the performance of quantum algorithms, error correction schemes, and noise-aware compilation techniques.

Noise models are tailored to capture the specific characteristics of quantum processors, such as native gate sets, connectivity constraints, and hardware-calibrated error rates. By incorporating these details, they offer a more accurate representation of physical systems, facilitating reliable benchmarking and optimization of quantum circuits. Depending on the use case, noise models range from simplified abstractions (e.g., depolarizing noise) to detailed, calibration-based models that closely reflect the actual hardware's noise profile.

The most common noise models used to evaluate the behavior of QECCs are:

1. **Phenomenological Noise Model:** This model assumes random Pauli errors (X, Y, or Z) on data qubits and measurement results at a physical error rate p. It simplifies noise modeling by treating syndrome extraction circuits as noiseless, apart from measurement errors, which makes it less realistic for real-world implementations [18].
2. **Circuit-Level Noise Model:** Extending the phenomenological model, the circuit-level model accounts for Pauli errors on all qubits during initialization, idle phases, and at every gate (including measurements) within a syndrome extraction circuit. Error rates may vary by operation

type (e.g., initialization, idle time, single- or two-qubit gates, measurement) but are assumed to remain independent of the specific qubits involved.

3. **Calibration-Based Noise Model:** This model provides a more accurate representation of noise but is computationally resource-intensive. It incorporates hardware-specific constraints, such as native gates and topology, and mimics the exact circuit executed on real devices. This model also considers error sources, including qubit parameters (relaxation and dephasing times, frequency, anharmonicity, and readout errors) and the specific error rates of native gates as provided by hardware vendors [19].

In this work, we employ the calibration-based noise model to simulate the effects of errors during quantum circuit execution. This model integrates device-specific error metrics such as two-qubit gate fidelities, single-qubit relaxation and dephasing times, and readout errors provided by the hardware vendor. By leveraging this detailed model, we can better analyze the impact of noise on quantum circuits and design techniques to mitigate these effects effectively.

2.2 Quantum circuit compilation process

The compilation process translates high-level quantum code into a hardware-executable format that respects the constraints of a quantum processor. This transformation adapts a quantum circuit into an equivalent one that can be executed on a quantum device while taking into account the hardware's specific limitations.

Similar to classical processors, quantum processors can execute only a predefined set of native instructions. Therefore, any high-level quantum circuit—designed to describe a quantum algorithm or application—must be compiled into a form that is compatible with the target quantum device. Crucially, the compiled quantum circuit must utilize only the native gate set of the device. For instance, the native gates of `ibm_sherbrooke` include the Echoed Cross-Resonance (ECR) gate, ID (identity), RZ (rotation around the z-axis), SX (square root of X), and X (Pauli-X) gates.

Additionally, quantum processors often have limited connectivity between qubits. As a result, gates can only be applied to qubits that are physically connected on the hardware. For quantum processors with local interaction constraints, this limitation becomes particularly evident when performing two-qubit (2Q) gates. Specifically, 2Q gates can only be applied to qubits that are directly connected via the device's topology. To perform a 2Q gate on qubits that are not directly connected—referred to as long-distance qubit interactions—SWAP[1] gates are introduced. SWAP gates are used to physically move the quantum information of the involved qubits closer together in the hardware's topology, enabling the execution of the desired two-qubit operation.

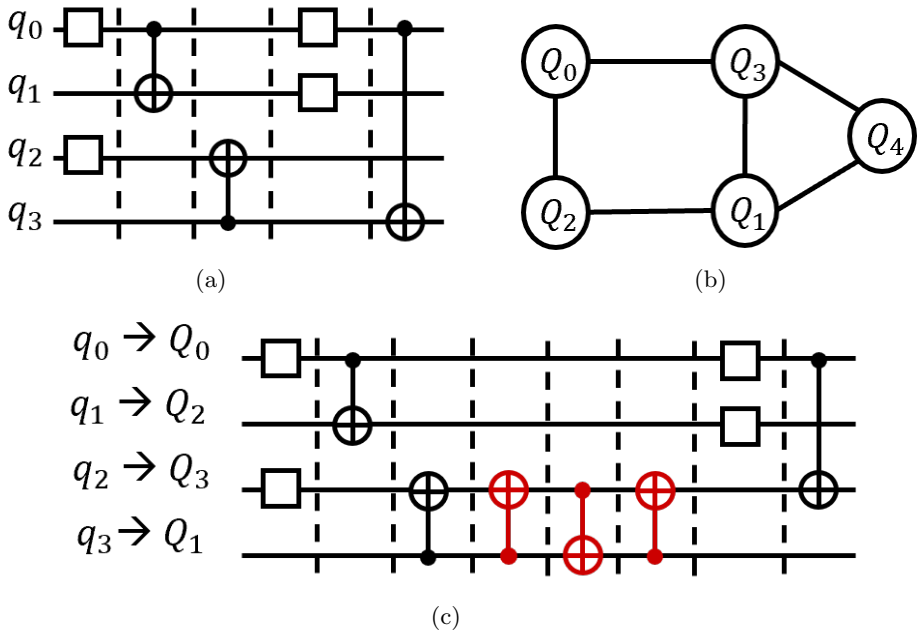
Figure 1 provides an example of the compilation process through several key steps. Consider the circuit in Fig. 1a – the original circuit – designed for execution on a quantum device with the topology shown in Fig. 1b. In this topology, physical qubits are represented as circles labeled Q_n (where n is the qubit index), and the edges indicate connections for performing 2Q gates. The initial qubit placement maps each virtual qubit in the circuit (q_n) to a corresponding physical qubit on the device ($q_n \rightarrow Q_n$), as shown in Fig. 1c. This mapping allows the first two CNOT gates to be executed since edges exist between the respective qubits (Q_0 and Q_2, Q_1 and Q_3). However, executing the third CNOT gate (between q_0 and q_3) requires moving the qubits involved to adjacent positions. This is achieved by introducing a SWAP gate, as shown in Fig.1c, that exchanges the state of Q_3 and Q_1.

Additional compilation steps, not depicted in this example, include i) operation scheduling, which exploits the parallelism of the circuit, and ii) gate nativization, which translates gates into the device's native gate set.

2.3 Noise-aware compilation

Noise-aware compilation optimizes the execution of quantum circuits by incorporating the noise characteristics of the quantum processor into the compilation process. This approach strategically places qubits with higher fidelities and routes operations in ways that minimize the impact of errors caused by factors such as decoherence, crosstalk, or gate inaccuracies.

[1]For two qubits A and B, SWAP(A,B) := {CNOT A,B; CNOT B,A; CNOT A,B;}.



(a) (b)

(c)

Figure 1. Quantum circuit compilation example: (a) original circuit, (b) backend topology, and (c) initial qubit mapping and routing result with the added SWAP gate (in red).

In this work, we focus on two primary noise sources: 2Q gate errors and readout errors, as these typically contribute the most to the overall error rates in current quantum processors. These noise metrics are used to guide decisions regarding qubit placement and routing, thereby improving the reliability and performance of quantum circuits on noisy hardware. We exclude single-qubit gate errors from our analysis, as their magnitudes are generally an order of magnitude smaller than those of 2Q gate errors and readout operations.

Noise characteristics are usually provided by hardware vendors, either through their APIs or as accessible data on their websites. In this work, we use IBM quantum processors, for which noise data is publicly available on the IBM Quantum website [20] and can also be accessed programmatically via Qiskit [21].

2.4 Compilation framework

As quantum processors scale in qubit count, the challenge of noise-aware compilation grows due to the increasing computational cost, particularly because initial qubit placement and routing are NP-hard problems [22]. To address this, we combined two state-of-the-art approaches: SABRE [23], for non-noise-aware qubit placement, and TriQ [24], for noise-aware qubit routing. These two techniques are discussed in detail below.

2.4.1 Initial qubit placement

The initial qubit placement refers to assigning logical qubits in the quantum circuit to physical qubits on the device. This step is critical because the placement of qubits can significantly impact the number of SWAP gates required during execution, which directly affects circuit fidelity and execution time.

We employed SABRE, a heuristic-based algorithm designed to iteratively optimize initial qubit placement. SABRE reduces the gate execution cost by dynamically considering the hardware topology and reordering gates to minimize the number of SWAP operations required later. While SABRE itself is not noise-aware, it serves as an efficient starting point, ensuring a reasonable initial placement before incorporating noise information in subsequent steps. This optimized initial mapping is then used as input for the noise-aware TriQ routing process, ensuring a more efficient overall compilation.

2.4.2 Qubit routing

In quantum processors, only nearest-neighbor interactions between qubits are allowed. Qubit routing ensures that interactions required by quantum polar codes, which often involve distant qubits, can be executed. This process introduces SWAP gates to move non-adjacent qubits into physically adjacent positions, enabling two-qubit gate operations.

For the routing stage, we utilized and extended TriQ, a noise-aware routing algorithm. TriQ optimizes qubit routing by incorporating the noise characteristics of the quantum processor—such as two-qubit gate errors and readout errors—into its decision-making process. This noise-aware approach minimizes the overall impact of noise on circuit fidelity.

In our implementation, we enhanced TriQ to ensure compatibility with the latest versions of Qiskit and extended its functionality. Notably, we introduced mid-circuit measurement capabilities to enable syndrome measurements. Also, we added an option to use the Floyd-Warshall algorithm for routing, which, in certain cases, reduces error accumulation [25].

2.5 Performance metrics

The effectiveness of the noise-aware compilation technique for mapping polar codes into noisy and restricted-connectivity quantum processors is quantified using three primary metrics: *preparation rate*, *relative gain*, and *logical error rate*.

2.5.1 Preparation rate

The **preparation rate** (p_{prep}) [16] measures the success rate of state preparation, specifically the proportion of prepared states that are accepted. Note that a successfully prepared state is still prone to errors, as some errors may go undetected by the error detection mechanism.

It is calculated as the ratio of the number of successful preparations (n_{prep}) out of the number of independent preparation attempts (n_{tot}):

$$p_{prep} = \frac{n_{prep}}{n_{tot}}$$

2.5.2 Logical error rate

The **logical error rate** (LER) quantifies the proportion of accepted states after the error detection phase that can be corrected, meaning that it measures the rate of undetected error. It is defined as the ratio of the number of correctable accepted states after the error detection phase (c) compared to the total number of accepted states after the detection phase (t).

$$LER = \frac{n_{corr}}{n_{prep}}$$

2.5.3 Relative gain

Relative gain (G) quantifies the improvement of preparation rate [see Sec. 2.5.1] achieved by the proposed method (S-TriQ) compared to the benchmark (Qiskit-3).

Let p_{prep_1} represent the preparation rate of S-TriQ, and p_{prep_0} denote the preparation rate of Qiskit-3. The relative gain is defined as:

$$G = \frac{p_{prep_1} - p_{prep_0}}{p_{prep_1}}$$

This metric can be interpreted in two ways:

1. **Time Savings**: By reducing the number of repetitions required to achieve successful state preparation.

2. **Space Savings**: By minimizing the total number of qubits required for the procedure, depending on the fault-tolerant preparation strategy being utilized.

2.6 Example of the workflow for a quantum polar code

In this example, we demonstrate the workflow of our noise-aware compilation technique for a quantum polar code. The specific input circuit and backend details are as follows:

- **Input Circuit**: Quantum Polar Code with N=8 for logical $|0\rangle$, requiring a total of 12 qubits—8 data qubits and 4 ancilla qubits. See Fig. 2 for the original circuit.
- **Target Backend**: IBM's superconducting quantum processor, `ibm_sherbrooke`, with a scale factor applied to the physical error rate that ranges from 0.1 to 1.

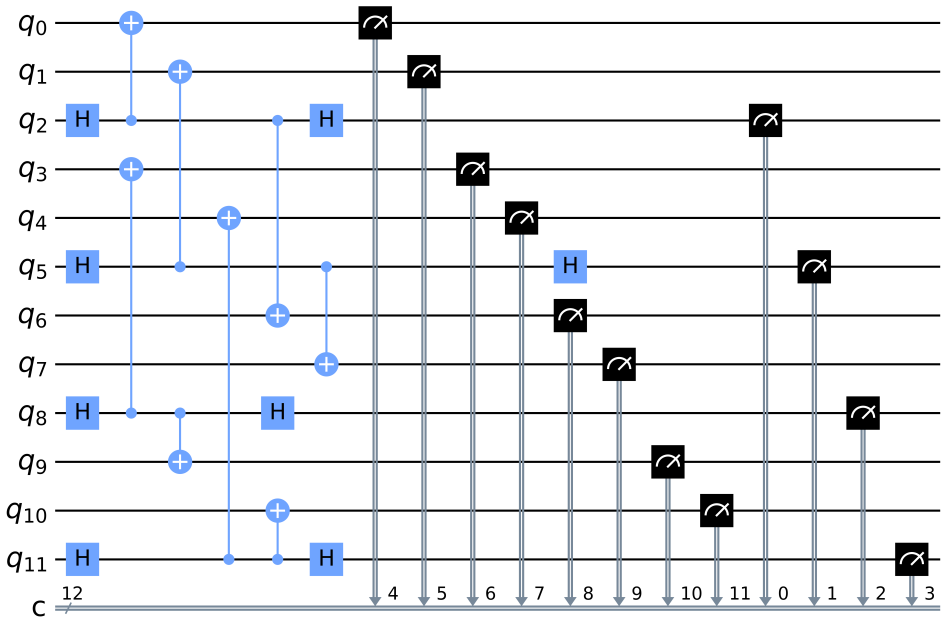


Figure 2. Measurement-based preparation of the logical $|0\rangle$ state for the $Q_1(N = 8, i = 3)$ code, where i indicates the information position in the polar code.

The workflow proceeds as follows:

1. **Processing Calibration Data**: Acquire the latest calibration data (two-qubits gate and readout error), and process it into a matrix to guide the noise-aware compilation.
2. **Initial Qubit Placement with SABRE**: Compile the circuit using the SABRE algorithm to determine the initial qubit mapping. In this example, the initial mapping is as follows: [$q_0 \to Q_{110}$, $q_1 \to Q_{117}$, $q_2 \to Q_{118}$, $q_3 \to Q_{100}$, $q_4 \to Q_{126}$, $q_5 \to Q_{116}$, $q_6 \to Q_{119}$, $q_7 \to Q_{115}$, $q_8 \to Q_{99}$, $q_9 \to Q_{98}$, $q_{10} \to Q_{108}$, $q_{11} \to Q_{112}$] This mapping corresponds to the order of virtual qubits assigned to physical qubits. Figure 3 illustrates this initial mapping: the top part of the figure shows a section of the `ibm_sherbrooke` topology, where numbers indicate the indices of the physical qubits, while the bottom part depicts how the virtual qubits are mapped onto the physical qubits, with the numbers representing the indices of the virtual qubits.
3. **Noise-Aware Qubit Routing with TriQ**: Use the TriQ algorithm to optimize routing based on noise characteristics, minimizing errors from two-qubit gates and readout operations. See Fig. 4 as the compiled circuit with the natives' gate from the selected device.
4. **Execution and Error Detection**: Run the compiled circuit. Apply the error detection gadget discarding states on which errors are detected. Pass successfully prepared states to the error correction decoder.
5. **Performance evaluation**: Evaluate the "Preparation Rate" and "Logical Error Rate" defined in Sec. 2.5

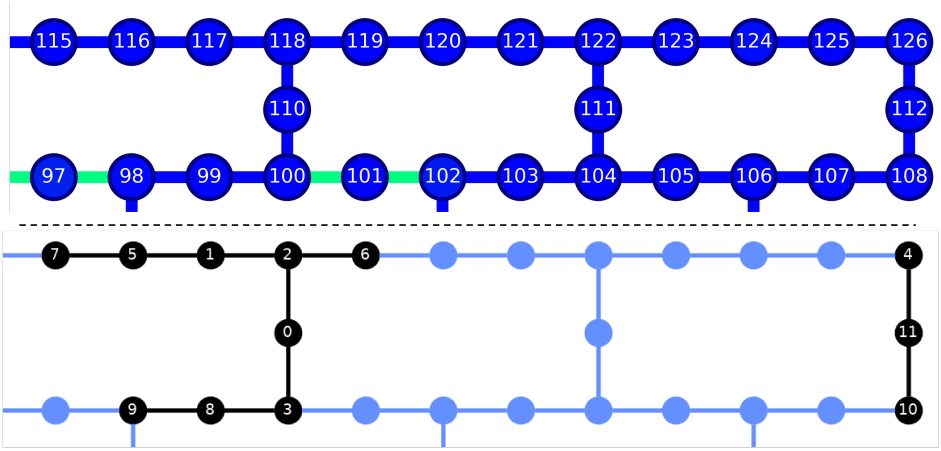


Figure 3. (Top) a part of `ibm_sherbrooke` topology used in this example, where numbers indicate the indices of the physical qubits. (Bottom) the initial mapping used in this example, with the numbers representing the indices of the virtual qubits

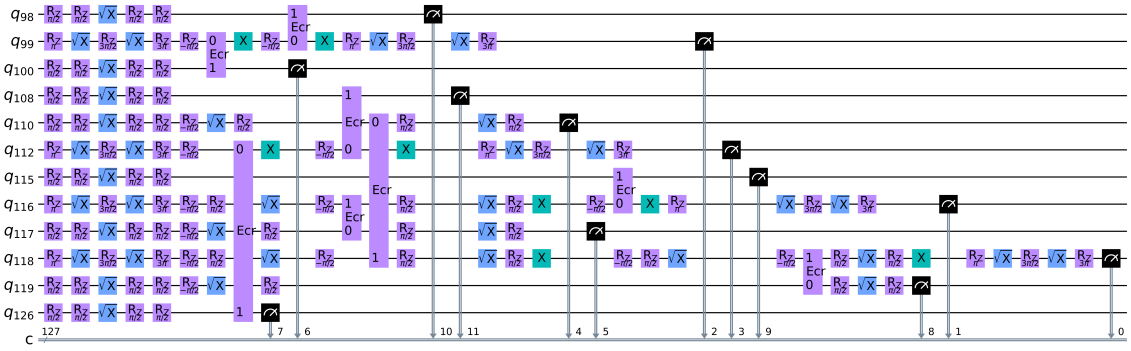


Figure 4. Compiled circuit with S-TriQ with the set of instructions based on the natives' gate from the selected device.

2.7 Repository of the implementation

The codes supporting this study's findings are openly available in the GitHub repository `https://github.com/HandyKurniawan/na_polar_codes_framework`.

3 Additional Information from the Experiment

3.1 Preparation rate

Table 1 presents the relative gains of the noise-aware compilation technique compared to Qiskit-3. The noise-aware S-TriQ approach demonstrated significant improvements, with relative gains of up to 54% for `ibm_brisbane` and 81% for `ibm_sherbrooke`. These gains can be interpreted as either time savings—by reducing the number of repetitions required to achieve a successful state preparation—or as space savings—by minimizing the overall qubit count needed for the procedure, depending on the fault-tolerant preparation strategy employed. The higher performance on `ibm_sherbrooke` can be attributed to lower error per layered gate (EPLG) of 2.1% compared to 3%(`ibm_brisbane`) [20].

3.2 CNOT count

In Table 2, the average number of CNOT gates for the different codes and compilation methods is included. For longer codes, S-TriQ adds 34% more CNOTs because of the additional SWAP gates used to reach a more reliable path during the routing (10 SWAP gates more in the worst case).

Table 1. Logical $|+\rangle$ preparation rate, depending on the compilation technique, the quantum hardware, and the code length (N).

Comp.	N	ibm_brisbane		ibm_sherbrooke	
		Prep.(%)	**Gain(%)**	**Prep.(%)**	**Gain(%)**
Qiskit-3	4	67.43	-	70.84	-
S-TriQ	4	94.47	28.62	94.95	25.39
Qiskit-3	8	25.55	-	23.63	-
S-TriQ	8	56.05	54.41	55.69	57.56
Qiskit-3	16	13.37	-	5.74	-
S-TriQ	16	28.93	53.78	30.31	81.06

Table 2. Comparison of the total number of CNOT (CX) gates for Quantum Polar Codes compiled using non-noise-aware (Qiskit-3) and noise-aware (SABRE-TriQ) for the logical state $|+\rangle$, and with code length (N).

Log	N	**Qiskit-3**	**S-TriQ**
$\|+\rangle$	4	11	11
$\|+\rangle$	8	58	78
$\|+\rangle$	16	126	165

3.3 Compilation time

Fig. 5 compares the compilation time (in seconds) for the noise-aware SABRE-TriQ approach across various code lengths. The compilation time remains relatively stable up to a code length of 32, after which there is a significant increase at code length 64. This jump in time is primarily due to the computational limitations of the hardware used in this research, indicating the need for higher computational power to efficiently handle larger code lengths.

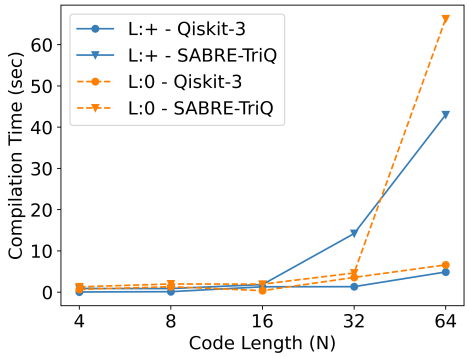


Figure 5. Average compilation time (in seconds) for quantum polar codes using noise-aware compilation, categorized by state and code length. The blue lines represent the logical $|+\rangle$ state, and the orange lines represent the logical $|0\rangle$ state. Solid and dashed lines correspond to Qiskit-3 and SABRE-TriQ compilation methods, respectively.

3.4 Decoding time

Fig. 6 illustrates the decoding time (in microseconds) for one accepted state. The results are promising, as the decoding time remains within hundreds of microseconds, suggesting that real-time performance is achievable even with non-embedded system software.

While hundreds of microseconds may seem far from the latency constraints typically observed in superconducting qubits, it is important to clarify that a real-time decoder does not necessarily need to meet such stringent latency limits of a round of measurements. For instance, in a recent paper by Google AI (Google Quantum AI and Collaborators. Quantum error correction below the surface code threshold. Nature (2024). https://doi.org/10.1038/s41586-024-08449-y), real-time decoding was

demonstrated with an average latency of 63 microseconds, despite a syndrome extraction cycle time of just 1.1 microseconds. Similar results and time budget (tens of microseconds) were reported by Quantum Machines in their recent work (https://arxiv.org/pdf/2412.00289).

Moreover, our current decoder implementation is in Python, which, while advantageous for rapid development and flexibility, is not the most time-optimized language. Implementing the decoder in a compiled language, such as C/C++ or RUST, could substantially reduce the latency. Furthermore, for even lower latencies, an FPGA or ASIC implementation would be a promising and scalable direction. As reported in the paper [26], an ASIC implementation achieved a latency of 630 nanoseconds for a classical code length of 1024, substantially longer than the target codes in our work. This illustrates that with hardware accelerators, we could significantly reduce the latency, potentially achieving nanosecond-level decoding.

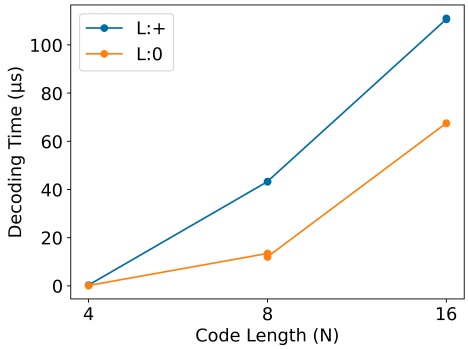


Figure 6. Average correction time per shot (in microseconds) for quantum polar codes using noise-aware compilation, categorized by state and code length. The blue line indicates the logical $|+\rangle$, and the orange line indicates the logical $|0\rangle$.

References

[1] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, no. 5886, pp. 802–803, 1982.

[2] J. Von Neumann, *Mathematical foundations of quantum mechanics: New edition*, vol. 53. Princeton University Press, 2018.

[3] A. Chatterjee, K. Phalak, and S. Ghosh, "Quantum error correction for dummies," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1, pp. 70–81, IEEE, 2023.

[4] D. Gottesman, *Stabilizer codes and quantum error correction*. California Institute of Technology, 1997.

[5] A. Y. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, no. 1, 2003.

[6] H. Bombin and M. A. Martin-Delgado, "Topological quantum distillation," *Physical review letters*, vol. 97, no. 18, p. 180501, 2006.

[7] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 86, no. 3, p. 032324, 2012.

[8] A. J. Landahl, J. T. Anderson, and P. R. Rice, "Fault-tolerant quantum computing with color codes," *arXiv:1108.5738*, 2011.

[9] A. M. Steane, "Active stabilization, quantum computation, and quantum state synthesis," *Physical Review Letters*, vol. 78, no. 11, p. 2252, 1997.

[10] A. M. Steane, "Fast fault-tolerant filtering of quantum codewords," *arXiv preprint quant-ph/0202036*, 2002.

[11] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[12] J. M. Renes, F. Dupuis, and R. Renner, "Efficient polar coding of quantum information," *Physical Review Letters*, vol. 109, no. 5, p. 050504, 2012.

[13] M. M. Wilde and S. Guha, "Polar codes for degradable quantum channels," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4718–4729, 2013.

[14] J. M. Renes and M. M. Wilde, "Polar codes for private and quantum communication over arbitrary channels," *IEEE Transactions on Information Theory*, vol. 60, no. 6, pp. 3090–3103, 2014.

[15] F. Dupuis, A. Goswami, M. Mhalla, and V. Savin, "Polarization of quantum channels using Clifford-based channel combining," *IEEE Trans. on Information Theory*, vol. 67, no. 5, 2021. arXiv:1904.04713.

[16] A. Goswami, M. Mhalla, and V. Savin, "Fault-tolerant preparation of quantum polar codes encoding one logical qubit," *Physical Review A*, vol. 108, no. 4, p. 042605, 2023.

[17] A. Goswami, M. Mhalla, and V. Savin, "Factory-based fault-tolerant preparation of quantum polar codes encoding one logical qubit," *Physical Review A*, vol. 110, no. 1, p. 012438, 2024.

[18] G. P. Gehér, C. McLauchlan, E. T. Campbell, A. E. Moylett, and O. Crawford, "Error-corrected hadamard gate simulated at the circuit level," *Quantum*, vol. 8, p. 1394, 2024.

[19] IBM, "Device backend noise model simulations," 2024. Available: https://qiskit.github.io/qiskit-aer/tutorials/2_device_noise_simulation.html (Accessed Oct. 1, 2024).

[20] IBM, "IBM Quantum Processors," 2024. Available: https://quantum.ibm.com/services/resources (Accessed Sep. 23, 2024).

[21] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024.

[22] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 international symposium on code generation and optimization*, pp. 113–125, 2018.

[23] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Proc. of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1001–1014, 2019.

[24] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Full-stack, real-system quantum computer studies: Architectural comparisons and design insights," in *Proc. of the 46th International Symposium on Computer Architecture*, pp. 527–540, 2019.

[25] A. E. Mirino *et al.*, "Best routes selection using Dijkstra and Floyd-Warshall algorithm," in *2017 11th International Conference on Information & Communication Technology and System (ICTS)*, pp. 155–158, IEEE, 2017.

[26] P. Giard, A. Balatsoukas-Stimming, T. C. Müller, A. Bonetti, C. Thibeault, W. J. Gross, P. Flatresse, and A. Burg, "PolarBear: A 28-nm FD-SOI ASIC for decoding of polar codes," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 7, no. 4, pp. 616–629, 2017.