Optimized Clifford Noise Reduction: Theory, Simulations and Experiments

Edwin Tham and Nicolas Delfosse

IonQ Inc.

5th August 2025

We propose several optimizations of the CliNR partial error correction scheme which implements Clifford circuits by consuming a resource state. Errors are corrected by measuring a sequence of Pauli operators that we refer to as the verification sequence. We first propose a global optimization algorithm searching for a verification sequence resulting in a low logical error rate using tabu search. Then, we introduce a proxy for the logical error rate which is easier to evaluate and we design a two-step optimization algorithm. First, a verification sequence minimizing the proxy is computed, then this sequence is refined by reintroducing the logical error rate. Finally, we identify a large group of automorphisms of the search space which preserve the proxy and we use this symmetry to reduce the size of the search space. This results in a $168\times$ (respectively $20,160\times$) reduction of the size of the search space for the optimization of verification sequences with three (respectively four) Pauli operators. Our numerical simulations for 20-qubit Clifford circuits with size 400 under the ion chain model show that our optimization algorithms improve the performance of CliNR by 25% and that the two-step optimization achieves the same results as the global optimization with 64% fewer evaluations of the logical error rate. Finally, we perform experiments on a 36-qubit trapped ion quantum computer, without mid-circuit measurements, showing that the CZNR variant of CliNR is at breakeven.

1 Introduction

Large-scale quantum computing applications likely require a fault-tolerant quantum computer in order to correct faults occurring during the computation before they spread to all the qubits [18, 21]. However, fault tolerance comes with a massive overhead in term of qubit and gate count.

Partial error correction is likely to play an important role in the near-term as it consumes fewer qubits and fewer gates. For example, the recent CliNR scheme has a qubit overhead of only about $3\times$ whereas standard quantum error correction codes generally consume tens or hundreds of physical qubits per logical qubit [9]. Even though it is not designed to be fault-tolerant, and it does not reduce the logical error rate as much as fault-tolerant quantum error correction, previous simulations show that this scheme could be useful to reduce the logical error rate of quantum circuits in near-term noise regimes.

With CliNR, the Clifford circuit to implement is decomposed into a sequence of t subcircuits and each subcircuit C is implemented in three steps as follows.

1. **Resource state preparation.** A resource state $|\psi\rangle$ is prepared on ancilla qubits. This state a stabilizer state [13].

2. **Resource state verification.** The resource state $|\psi\rangle$ is tested by measuring some of its stabilizers. By definition, these measurements should return a trivial outcome. If not, we know that a fault has occurred, whereupon we reset the qubits and restart the preparation of $|\psi\rangle$.

3. **Resource state injection.** If all measurement outcomes are trivial, the subcircuit C is applied to an n-qubit input state by consuming the resource state $|\psi\rangle$.

This approach to implement quantum gates through resources state has been extensively used before CliNR. Shor used resource states to perform error correction and logical Toffoli gates in his seminal paper introducing fault-tolerant quantum computation [21]. Steane and Knill proposed quantum error correction protocols based on different resource states [16, 22]. Magic states are resource states that can be consumed to implement non-Clifford gates [3]. Gate teleportation provides a general framework to this approach and it is a core ingredient in CliNR [15]. Measurement-based quantum computation [5, 19] and fusion-based quantum computation [1] also rely on resource states to perform quan-

tum error correction and fault-tolerant logical operations. These schemes generally require a long sequence of stabilizer measurements for resource state verification to achieve fault tolerance or to reach the full code distance. Here, we focus on CliNR and our goal to improve the logical error rate of CliNR implementation of a circuit using a relatively short sequence of stabilizer measurements, which keeps the overhead low.

We refer to the stabilizers selected for the resource state verification as the *verification sequence*. Our main goal is to address the following problem.

The CliNR optimization problem. *Given a noise model and a Clifford circuit C, find a verification sequence that minimizes the logical error rate of the CliNR implementation of C.*

The main challenge with the CliNR optimization problem is the combinatorial explosion of the number of possible verification sequences. The number of choices for a n-qubit circuit is

$$\prod_{i=0}^{r-1} \left(2^{2n} - 2^i\right) \tag{1}$$

where r is the verification sequence size. Therein, we assume that the r stabilizers of the verification sequence are independent. For $n = 10$ and $r = 2$ this number is already larger than 10^{12} and it reaches more than 10^{36} when $n = 20$ and $r = 3$ which is a relevant regime for CliNR based on previous simulations.

If the circuit is sufficiently small, we can enumerate all possible verification sequences. Each verification sequence can then be evaluated through Monte-Carlo simulation estimates of the logical error rate of C under the corresponding CliNR implementation. This provides a solution to the CliNR optimization problem; though, this exhaustive approach is clearly limited to small circuits.

In [9], the verification sequence was designed either by selecting stabilizers uniformly at random or by selecting low-weight stabilizers because they can be implemented with fewer gates. It is natural to expect that a more careful selection of the stabilizers, tailored to the exact noise model and structure of the circuit, would lead to a reduction of the logical error rate achieved with CliNR.

To go beyond these naive optimizations, we introduce a heuristic optimization algorithm (Algorithm 1) based on tabu search [11, 12]. We chose tabu search for its simplicity, but this subroutine can be replaced by other metaheuristic optimization algorithm such as hill climbing, simulated annealing or more involved strategies. We start from a randomly selected verification sequence. Then, at each iteration we evaluate a set of neighboring verification sequences by replacing one of the stabilizers within that sequence by a random stabilizer of the resource state. The neighboring verification sequence that reduces the logical error rate the most is accepted and added to a tabu list – this prevents the same verification sequence from being revisited over a set number of iterations, and helps the algorithm avoid local minima.

The most resource intensive subroutine in Algorithm 1 is the cost function evaluation, that is the estimation of the logical error rate of CliNR for a given verification sequence. To speed up this subroutine, we introduce a proxy cost function, that is an approximation of the logical error rate and we propose a two-step optimization algorithm (Algorithm 3). First, we search for a verification sequence that minimizes our proxy. Then, in a second step, we refine our verification sequence by introducing the actual cost function.

Our proxy has two features that allow us to speed up the exploration of the search space. First, it is easier to evaluate than the actual cost function. Second, it is invariant under the action of a large group of automorphisms of the search space. This allows us to reduce the size of the search space by identifying points that are equivalent up to the action of this group. In other words, we reduce the search space by identifying verification sequences which have the same proxy cost.

Interestingly, the resulting reduced search space is a well-studied object in geometry, topology and combinatorics called the *Grassmann manifold*, which has applications in machine learning, computer vision or imagine processing [2]. In what follows, we use the term *Grassmann graph* because we are working with finite fields and then the Grassmann manifold has a graph structure.

We used our two optimization algorithms to improve the CliNR implementation of random Clifford circuits. We perform numerical simulations for random n-qubit Clifford circuits with size $s = n^{1.8}$ and $s = n^2$ with $n = 20$ under the ion chain model proposed recently as a model for long chains of trapped ions [25]. For $s = n^2$, the previous CliNR scheme based on a random verification achieves a 37% reduction in logical error rate and the CliNR optimization algorithms proposed in this work achieve an additional 25% reduction. Moreover, the two-step optimization algorithm (Algorithm 3) reaches the same logical error rate as the global optimization algorithm (Algorithm 1) using 64% fewer evaluations of the logical error rate. This reduction in the number of evaluations of the logical error rate is particularly relevant for practical applications where optimizing CliNR may include estimating the logical error rate by executing a program on a real machine.

Finally, we performed experimental demonstrations of the CZNR variant [9] of CliNR that implements cir-

cuits made with CZ gates, showing that the scheme is at breakeven. These experiments are executed on a IonQ Forte-Enterprise quantum computer with 36 qubits encoded in a long chain of Ytterbium-171 ions. This machine is not equipped with mid-circuit measurements, therefore we keep the ancilla qubits used for stabilizer measurements idle until the end of the circuit. We expect that a proper implementation of CZNR with mid-circuit measurements will lead to a further reduction of the logical error rate as it would remove the need to keep qubits idle until the final measurement.

We compare the logical error rate of the direct implementation and the CZNR implementation of 10-qubit circuits with 90 CZ gates and we conclude that our CZNR experiment achieves breakeven performance based on three following observations. (i) The direct implementation and the CZNR implementation achieve approximately the same logical error rate. (ii) Increasing the length of the verification sequence keeps the logical error rate constant. That means that noise introduced by implementing the verification sequence is compensated by the noise it removes. (iii) Increasing the noise rate in the resource state leads to an increase in the logical error rate. Observations (i) and (ii) are not enough by themselves to show that the scheme is at breakeven. Indeed, one may fulfill these two with a system saturated with noise, *i.e.* such the logical error rate remains constant because it has reached a maximum value. Item (iii) excludes this explanation. All together, these observations suggest that our CZNR experiment is at breakeven. A small improvement in gate fidelity may allow us to reach the below-threshold regime where the CZNR implementation has a lower logical error rate than the direct implementation.

The CliNR scheme is related to the coherent parity check (CPC) scheme [7, 17, 20, 24], which was demonstrated experimentally recently [24]. The main advantage of CliNR is that it avoids the exponential blowup in shot count of the CPC scheme. This is because the Clifford circuit we wish to implement is partitioned into smaller sub-circuits that are implemented through resource states as described above. Our experiment illustrates this feature with a partition of the circuit into two subcircuits.

In the remainder of the paper, Section 2 reviews the background and notations and Section 3 provides a detailed description of the CliNR optimization problem. Our first algorithm that provides an approximate solution of this problem is the global optimization algorithm (Algorithm 1) described in Section 4. The goal of Section 5 is to propose a two-step optimization algorithm based on a proxy for the cost function. The key theoretical result justifying this algorithm is Proposition 1 which identifies a group of automorphisms of the search space that preserves our proxy. This proposition also shows that the reduced search space, obtained by identifying the points living in the same orbits under the group action, is the Grassmann graph. Then, Algorithm 3 describes the two-step CliNR optimization algorithm which uses the proxy as an intermediate optimization cost function. This intermediate step is described in Algorithm 2 which is executed in the Grassmann graph instead of the original search space. Section 6 discusses numerical results comparing our two optimization algorithms. Section 7 presents experimental results on a 36-qubit trapped ion quantum computer showing breakeven performance.

2 Background

In what follows, $n \geq 1$ is an integer and $\mathcal{P}_n$ denotes the n-qubit *Pauli group* quotiented by the phase subgroup $\{\pm I, \pm iI\}$. Its elements are called *Pauli operators*. For any vector $u = (u_1, \ldots, u_n) \in \mathbb{Z}_2^n$, denote by $X^u = X_1^{u_1} \ldots X_n^{u_n}$ the operator of $\mathcal{P}_n$ acting as X on the support of u. The operator Z^u is defined similarly. Any operator $P \in \mathcal{P}_n$ is of the form $P = X^u Z^v$ with $u, v \in \mathbb{Z}_2^n$ because its phase can be ignored in the quotiented Pauli group.

Given two Pauli operators $P = X^u Z^v$ and $P' = X^{u'} Z^{v'}$ in $\mathcal{P}_n$, denote

$$[P, Q] = (u|v') + (u'|v) \pmod 2 \tag{2}$$

where $(x|y) = \sum_{i=1}^n x_i y_i \pmod 2$ is the binary inner product. The map $[\cdot, \cdot]$ is a symmetric bilinear map, meaning that for all P, Q in $\mathcal{P}_n$, we have $[P, Q] = [Q, P]$, and for all $P, P', Q, Q' \in \mathcal{P}_n$, we have

$$[P, QQ'] = [P, Q] + [P, Q'] \tag{3}$$
$$[PP', Q] = [P, Q] + [P', Q] \tag{4}$$

where these sums are taken modulo 2.

We have $[P, Q] = 0$ iff the operators P and Q commute and $[P, Q] = 1$ iff they anti-commute. Given a subset $S \subseteq \mathcal{P}_n$ of Pauli operators, the set of Pauli operators in $\mathcal{P}_n$ that commute with all the operators of S is denoted by $S^\perp$. If S generates a rank-r subgroup of $\mathcal{P}_n$, then $S^\perp$ is a subgroup of $\mathcal{P}_n$ with rank $n - r$.

A *Clifford gate* is a unitary gate that maps any Pauli operator onto a Pauli operator by conjugation. Mathematically, a n-qubit unitary operation U is a Clifford gate if for all $P \in \mathcal{P}_n$, we have $UPU^{-1} \in \mathcal{P}_n$. A *Clifford circuit* is a sequence of Clifford gates.

A n-qubit *stabilizer state* is a n-qubit state that is fixed by n independent commuting Pauli operators. Any Pauli operator that fixes a stabilizer state is said to be a *stabilizer* of this state. It is easy to check that Clifford circuits map stabilizer states onto stabilizer states.

As a result, applying a Clifford circuit to a register of qubits initialized in a state $|0\rangle$ produces a stabilizer state.

We focus on circuits made with preparation of a qubit in the state $|0\rangle$, unitary single-qubit and two-qubit gates and measurements. For simplicity, we assume that the unitary gates are implemented sequentially.

We consider the standard *circuit-level noise* model. Noise occurring during a preparation, idle step and unitary operation is represented by a depolarizing channel acting on its support, inserted right after the operation. Measurement noise is modeled as a bit-flip of the measurement outcome. The depolarizing noise rate and the bit-flip rate may depend on the corresponding operation.

Any Pauli error occurring during a n-qubit circuit made with qubit preparations, Clifford gates and measurements is equivalent to a Pauli error occurring at the end of the circuit, that we refer to as the *output error*. The output error can be computed efficiently by conjugating the initial error through the Clifford gates of the circuit [14]. This defines a probability distribution over the set $\mathcal{P}_n$ of all possible output errors, that we call the *output noise distribution* of the circuit.

3 Detailed description of the optimization problem

The CliNR scheme implements a n-qubit circuit C using a *resource state* $C_R|\Phi_n\rangle$, where

$$|\Phi_n\rangle = 2^{-n/2} \sum_{x\in\mathbb{Z}_2^n} |x\rangle \otimes |x\rangle \tag{5}$$

is the state of n Bell pairs supported on qubits i and $n+i$ for $i = 1, \ldots, n$. The resource state preparation circuit could be $C_R = I \otimes C$ or, alternatively $C_R = C_1^{-1} \otimes C_2$ with $C = C_1 C_2$.

We assume that the stabilizers of the verification are measured sequentially. A stabilizer $P_{i_1} \ldots P_{i_w} \in \mathcal{P}_n$ acting on w qubits is measured using a single ancilla qubit prepared in the $|+\rangle$-state and a sequence of controlled Pauli gates applying $CP_{i_1} \ldots CP_{i_w}$ controlled on the ancilla qubit. The outcome is extracted by measuring the ancilla qubit in the X basis.

The *verification sequence* V is defined to be an ordered r-tuple of stabilizers of the resource state $C_R|\Phi_n\rangle$. The CliNR implementation of a circuit C using the verification sequence V is denoted $\mathrm{CliNR}(C_R, V)$. As an example, Fig. 1 shows the CliNR implementation of the circuit $C = H_1 CX_{1,2} CZ_{1,3}$ with $C_R = I \otimes C$ using the verification sequence $V = XIIZII, IZIZZI$.

The *logical error rate* of the CliNR implementation of a Clifford circuit C is defined to be the probability that the output error is non-trivial, *i.e.* $1 - \mathbb{P}(I)$ where $\mathbb{P}$ is the output noise distribution of the CliNR implementation of C.

The CliNR optimization problem takes as an input a resource state preparation circuit C_R for a Clifford circuit C and the output is a verification sequence V minimizing the logical error rate of $\mathrm{CliNR}(C_R, V)$. In what follows, the logical error rate of $\mathrm{CliNR}(C_R, V)$ is denoted $p_{\log}(\mathrm{CliNR}(C_R, V))$.

We optimize the implementation of a single CliNR subcircuit, although CliNR may split the input circuit into subcircuits, because the verification sequence associated with each subcircuit can be optimized independently. To further simplify the problem, we can fix the size r of the verification sequence because it is typically small (the proof of Theorem 2 in [9] uses $r = O(\log n)$ for Haar random Clifford circuits acting on n qubits.). Therefore, we can run over all r up to a bound B, find an optimized sequence for each r, and then select the best verification sequence with $r \leq B$.

4 Global CliNR optimization

This section describes an algorithm for the CliNR optimization problem based on tabu search [11, 12]. We use tabu search with a random candidate list strategy. The pseudocode is provided in Algorithm 1.

The verification sequence is initialized with r stabilizers of the resource state selected uniformly at random. Then, we proceed over $i_{\max}$ iterations. At each iteration, a set of m candidate verification sequences is selected in the neighborhood of the current best verification sequence V_{opt}. These candidates are obtained by replacing a stabilizer of V_{opt} by a random stabilizer of the resource state. After rejecting any sequence that exists in a tabu list, the CliNR logical error rate of the candidate verification sequences is estimated using Monte-Carlo simulations. The verification sequence V_{opt} is updated if a lower logical error rate is reached by a candidate verification sequences and this verification sequence is added to the tabu list. The tabu list is kept to a fixed length ℓ by removing entries in first-in-first-out (FIFO) order.

Other initializations of this starting verification sequence and different choices for the sequence update are possible. One could for instance favor low-weight operators. The parameters ℓ, m and $i_{\max}$ can be chosen through grid search, or any other hyperparameter optimization procedure.

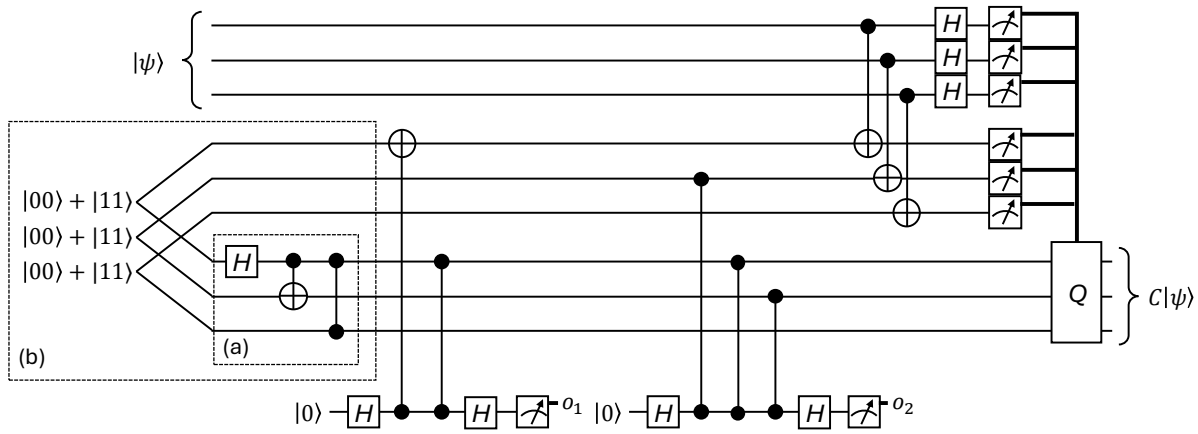

Figure 1: The CliNR implementation of the circuit $C = H_1 CX_{1,2} CZ_{1,3}$. The operation Q is a Pauli operator that depends on the measurement outcomes. The circuit C appears in (a). The resource state is prepared in (b) by applying the circuit $C_R = I \otimes C$ to n Bell states. This CliNR implementation uses the verification sequence $V = (XIIZII, IZIZZI)$. The measurement outcome o_1 is the outcome of $V_1 = XIIZII$ and o_2 is the outcome of $V_2 = IZIZZI$. If one of these outcomes is non-trivial, the circuit restarts from the beginning of (b).

Algorithm 1: Global CliNR optimization

Input: A circuit-level noise model. A resource state preparation circuit C_R. The stabilizer group S of the resource state $C_R|\Phi_n\rangle$. A positive integer r. Optimization parameters ℓ, m, and $i_{\max}$.

Output: A verification sequence.

Initialize a verification sequence V_{opt} with r stabilizers selected uniformly at random in S.
Initialize an empty tabu list L_{tabu}.
for $i \leftarrow 0$ **to** $i_{\max} - 1$ **do**
  Initialize an empty candidate list L_{cand}.
  Select $j \in [1, r]$ uniformly at random.
  for $k \leftarrow 0$ **to** $m - 1$ **do**
    Initialize $V = V_{\text{opt}}$.
    Replace V_j by a stabilizer selected uniformly at random in S.
    Add V to L_{cand}.
  for V *in* L_{cand} **do**
    if V *is not in* L_{tabu} **then**
      Estimate the logical error rate $p_V = p_{\log}(\text{CliNR}(C_R, V))$.
      if $p_V < p_{V_{\text{opt}}}$ **then**
        $V_{\text{opt}} \leftarrow V$.
  if V_{opt} *is not in* L_{tabu} **then**
    Append V_{opt} to L_{tabu}.
  if $|L_{\text{tabu}}| > \ell$ **then**
    Remove the first element of L_{tabu}.
return V_{opt}.

5 CliNR optimization through a proxy cost function

This section describes a two-step optimization algorithm based on a proxy cost function for the CliNR optimization problem. The proxy cost function is defined in Section 5.1. In Section 5.2, we identify a group of symmetries of the search space that keep the proxy cost function invariant and we construct the quotient of the search space by this group of symmetry. Section 5.3 describes an algorithm to optimize the proxy cost function in the quotiented search space. We refer to this algorithm as the proxy cost optimization. Our two-step CliNR optimization algorithm, described in Section 5.4, first executes the proxy cost optimization and then refine the choice of the verification sequence based on the actual cost function. The main advantage of this approach is that the proxy cost function can be evaluated without a Monte Carlo simulation and the quotiented search space is significantly smaller than the original search space. Our simulations presented in Section 6 shows that the two-step CliNR optimization produces better results than the global CliNR optimization.

5.1 Proxy cost function

The *cost function* in the CliNR optimization problem is the logical error rate $p_{\log}(\text{CliNR}(C_R, V))$.

To define our proxy cost function, consider the resource state preparation circuit. It is a sequence of s operations O_i with noise rate p_i in the circuit-level noise model introduced in Section 2. By definition of this noise model, each operation O_i is followed by a ran-

dom Pauli error E with probability p_i and E is selected uniformly at random in the set of non-trivial Pauli operators acting on the support of O_i. Define Ω_i to be the set of all possible Pauli operators obtained by propagating a Pauli error E that may occur right after O_i to the end of the circuit. The propagation of an error can be computed efficiently using Gottesman-Knill algorithm [14]. Stim is a convenient tool to perform this computation [10]. If the operations O_i acts on w qubits, then the set Ω_i contains $4^w - 1$ errors and each of them has probability $\tilde{p}_i := p_i/(4^w - 1)$.

With these notations, define the *proxy cost function* as

$$\mathrm{Proxy}(V) = \sum_{i=1}^{s} \sum_{P \in \Omega_i} \tilde{p}_i \delta_{\{P \in V^\perp \setminus S^\perp\}}. \tag{6}$$

Therein, the resource state preparation circuit is fixed and this proxy assigns a cost with each verification sequence V. The condition $P \in V^\perp$ means that P commutes with all the element of the verification sequence and therefore it is not detected by the verification sequence. The condition $P \notin S^\perp$ means that P has a non-trivial effect on the resource state and therefore induces a logical error. This proxy is a first order approximation of the logical error rate when no fault occurs during the resource state verification and injection.

Our motivation for introducing this proxy is twofold. First, it is easy to evaluate without a Monte-Carlo simulation. We precompute the sets Ω_i and we use them to evaluate the proxy cost $\mathrm{Proxy}(V)$ of a verification sequence V in a single pass. Second, it is invariant under a large group of symmetry which we describe next. These symmetries allow us to reduce the size of the search space.

5.2 Symmetry of the search space

The search space has a natural graph structure that we formally introduce now. We also review the definition of the Grassmann graph [2] and we establish a connection between the two spaces. Throughout this section, S is the stabilizer group of the resource state and r is the size of the verification sequence.

The *search graph*, denoted $\mathrm{Gs}(S, r)$, is defined to be the graph whose vertices are the verification sets V containing r independent stabilizers of S. Two vertices of $\mathrm{Gs}(S, r)$ are connected by an edge if they differ by a single stabilizer, *i.e.* V and V' are connected if $|\{i \in [1, r] \mid V_i \neq V'_i\}| = 1$. This definition is motivated by the fact that each modification of the verification in Algorithm 1 corresponds to a move from a vertex of the search graph to one of its neighbors.

The *Grassmann graph*, denoted $\mathrm{Gr}(S, r)$, is defined to be the graph whose vertices are the subgroups of S with rank r. Two vertices of $\mathrm{Gr}(S, r)$ are connected if the intersection of the corresponding subgroups is a subgroup with rank $r - 1$.

To establish a connection between the search graph and the Grassmann graph, we introduce the map

$$\alpha : GL_r(\mathbb{Z}_2) \times \mathcal{P}_n^r \longrightarrow \mathcal{P}_n^r \tag{7}$$

$$(M, V) \longmapsto V' \tag{8}$$

where $GL_r(\mathbb{Z}_2)$ is the group of invertible $r \times r$ matrices with binary coefficients. The image of (M, V) is the tuple $V' = (V'_1, \ldots, V'_r)$ with

$$V'_i = \prod_{j=1}^{r} V_j^{m_{i,j}} \tag{9}$$

where $M = (m_{i,j})_{1 \leq i,j \leq r}$.

The following proposition shows that the map α induces a group action on the vertex set of the search graph which allows us to quotient this vertex set producing a reduced search space. The proof of this result in included in Appendix A.

Proposition 1. *The map α induces a group action of the group $GL_r(\mathbb{Z}_2)$ onto the vertex set of the search graph $\mathrm{Gs}(S, r)$ which preserves the proxy cost function. Moreover, the quotient of the vertex set of the search graph by $GL_r(\mathbb{Z}_2)$ is the vertex set of the Grassmann graph $\mathrm{Gr}(S, r)$.*

The proof of this proposition relies on the fact that $\alpha(M, \cdot)$ is a bijection of the vertex set of the search graph. In general, it is not a graph automorphism because it does not preserve the edges of the search graph and the Grassmann graph is not the quotient of the search graph by $GL_r(\mathbb{Z}_2)$. We only show that there is a bijection between the vertex set the search graph quotiented by $GL_r(\mathbb{Z}_2)$ and the vertex set of the Grassmann graph. This bijection is sufficient for our purpose of reducing the size of the search space.

5.3 Proxy cost optimization

This section describes the proxy cost optimization which is the core subroutine of the two-step CliNR optimization algorithm. The proxy cost optimization exploits the symmetry of the search space demonstrated in Proposition 1.

Algorithm 2 follows closely the steps of Algorithm 1 with three major differences. First, the estimation of the logical error rate is replaced by the proxy cost function. Second, instead of exploring the search graph, Algorithm 2 explores the Grassmann graph to find a verification sequence that reaches a low value for the proxy cost function. Third, it returns a subgroup $\langle V \rangle$

instead of a verification sequence V. We refer to such a subgroup as a verification subgroup.

For our simulations in Section 6, we added the condition that candidate subgroups V have rank exactly r in Algorithm 2 to our implementation of this algorithm. To keep the pseudo code simple, we omit this in Algorithm 1 and Algorithm 2.

Algorithm 2: Proxy cost optimization

Input: A circuit-level noise model. A resource state preparation circuit C_R. The stabilizer group S of the resource state. A positive integer r. Optimization parameters ℓ, m, and $i_{\max}$.

Output: A verification subgroup.

Initialize a verification sequence V_{opt} with r stabilizers selected uniformly at random in S.
Initialize an empty tabu list L_{tabu}.
for $i \leftarrow 1$ **to** $i_{\max}$ **do**
  Initialize an empty candidate list L_{cand}.
  Select $r-1$ operators $V_1, \ldots, V_{r-1}$ in $\langle V_{\text{opt}} \rangle$ uniformly at random.
  for $j \leftarrow 1$ **to** m **do**
    Select a stabilizer V_r in $S \backslash \langle V_{\text{opt}} \rangle$ uniformly at random.
    Add $V = (V_1, \ldots, V_r)$ to L_{cand}.
  for V *in* L_{cand} **do**
    if V *is not in* L_{tabu} **then**
      Compute the proxy cost function $\text{Proxy}(V)$.
      if $\text{Proxy}(V) < \text{Proxy}(V_{\text{opt}})$ **then**
        $V_{\text{opt}} \leftarrow V$.
  if $\langle V \rangle$ *is not in* L_{tabu} **then**
    Append $\langle V \rangle$ to L_{tabu}.
  if $|L_{\text{tabu}}| > \ell$ **then**
    Remove the first element of L_{tabu}.
return $\langle V_{\text{opt}} \rangle$.

5.4 Two-step CliNR optimization

We now have all the ingredients to describe the two-step CliNR optimization algorithm.

Algorithm 2 returns a subgroup $\langle V \rangle$. Then, we extract a verification sequence by selecting elements from this subgroup. The resulting two-step optimization of CliNR is described in Algorithm 3.

When the verification subgroup is sufficiently small, the second step could be performed by selecting a verification sequence inside the verification subgroup that minimizes the actual cost function by exhaustive search.

If the verification subgroup is too large for exhaustive search, we performe the second step by executing Algorithm 1 to search over the verification sequence inside $\langle V \rangle$, that is replacing S by $\langle V \rangle$ in this algorithm. This results in Algorithm 3 which is generally faster than executing Algorithm 1 with S because r is typically small.

For simplicity, in Algorithm 3, we use the same values for the optimization parameters used when calling Algorithm 2 and Algorithm 1. Relaxing this assumption could help to further improve the performance of this algorithm.

Algorithm 3: Two-step CliNR optimization

Input: A circuit-level noise model. A resource state preparation circuit C_R. The stabilizer group S of the resource state. A positive integer r. Optimization parameters ℓ, m, and $i_{\max}$.

Output: A verification sequence.

Compute a verification subgroup $\langle V \rangle$ using Algorithm 2 with parameters ℓ, m, and $i_{\max}$.
Return a verification sequence obtained by running Algorithm 1 with $S = \langle V \rangle$ and with parameters ℓ, m, and $i_{\max}$.

5.5 Reduction of the search space

Proposition 1 highlights a key advantage our proxy cost function (together with its low-complexity evaluation). Its large group of symmetry allows us to reduce the size of the search space by identifying the points that differ in a symmetry preserving the proxy cost function. Proposition 1 proves that the search graph can then be replaced by the Grassmann graph whose number of vertices is given by

$$\frac{\prod_{i=0}^{r-1} \left(2^{2n} - 2^i\right)}{\prod_{i=0}^{r-1} \left(2^r - 2^i\right)}. \tag{10}$$

This number is significantly smaller than the number of vertices of the search graph computed in Eq. (1). For $r = 3$, this provides a $168\times$ reduction of the size search graph and for $r = 4$ the Grassmann graph is $20,160\times$ smaller than the search graph.

In practice, this reduction of the search space allows the two-step CliNR optimization algorithm to reach a better logical error rate than the global CliNR optimization algorithm using the same number of iterations as we observe in our numerical simulations.

6 Numerical Demonstration

In this section, we apply Algorithm 1 and Algorithm 3 to optimize the CliNR implementation of random Clifford circuits and we estimate their logical error rate

through numerical simulations. These numerical results show that the CliNR optimization algorithms proposed in this paper achieve a lower logical error rate than the standard version of the CliNR scheme [9].

We estimate the logical error rate of random Clifford circuits and their CliNR implementations under the *ion chain model* [25]. With this model unitary operations are implemented sequentially. Preparations, measurements and resets can be performed simultaneously on an arbitrary subset of qubits. Each operation is followed by depolarizing noise on the support of the operation with rate p for two-qubit gates, $p/10$ for single-qubit operations, and measurement outcomes are flipped with probability $p/10$. Noise on an idle qubit during an operation is represented by depolarizing noise with rate $p/100$, except if the qubit is idle during a measurement. Then, its noise rate is $\tau_m p/100$ for some constant τ_m, representing the measurement time, which we set to $\tau_m = 30$. In the simulations presented in this section, we use $p = 10^{-4}$.

In Fig. 2, we show simulation results for two randomly-chosen Clifford circuits C; each is chosen by first generating a n-qubit Haar-random Clifford circuit with n compiled into the $\{H, S, S^\dagger, CX\}$ gateset [4], and then truncating it to sizes $s = n^{1.8}$ and $s = n^2$ respectively. We then compare the logical error rate of C across four different implementations: (i) the direct implementation of C, that is without CliNR (horizontal dotted line), (ii) the CliNR implementation of C with a uniform random verification sequence with size r as original proposed in [9] (horizontal dot-dashed), (iii) the CliNR implementation of C with a verification sequence with size r optimized with Algorithm 1 (dashed curve), (iv) the CliNR implementation of C with a verification sequence with size r optimized with Algorithm 3 (continuous curve). These simulations are performed with $n = 20$ and $r = 4$ and for both (iii) and (iv) we use the values $\ell = 10$ and $m = 5$ for the parameters of Algorithms 1 and 3and the number of iterations $i_{\max}$ varies.

We plot the logical error rate of the CliNR implementation of C as a function of the number of evaluations of the logical error rate $p_{\log}$, up to a maximum of 500 evaluations (evaluations of the proxy cost function introduced in Section 5.1 constitutes a minuscule fraction of optimization, and is not included). Each curve represents an average over many repetitions of each respective circuit and optimization algorithm, while the shaded area around them indicates one-standard-error boundaries. For a given circuit C, implementation, and optimization procedure, we repeat that procedure identically, 250 times. Each evaluation of the logical error rate is done over $50,000$ Monte-Carlo shots. Further, to exclude trivial outcomes, we impose additional constraints not explicit specified in pseudocodes for Algorithms 1 and 3: that the rank of any subgroup $\langle V \rangle$ returned by Algorithm 2 have rank r, and that trivial entries $I^{\otimes n}$ cannot be admitted into a verification sequence V returned by Algorithm 1.

We observe that the optimization algorithms we proposed significantly improve the performance of CliNR. In the case of $s = n^2$, while random verification sequences yield an average of $\approx 37\%$ reduction in noise over direct implementation, our global (and two-step) optimization yields a further $\approx 21\%$ (25%) reduction in logical error rate. More notably, to achieve the logical error rate reached by global optimization across the full stabilizer group with 500 evaluations of the cost function, the two-step algorithm required on average only 180 evaluations.

A similar result is observed in the case of $s = n^{1.8}$. However, especially noteworthy in that case, is that whereas CliNR with random verification sequences fail to outperform direct implementation, our optimizations yield improvements over the latter of $\approx 17\%$ and $\approx 20\%$ respectively. This illustrates a case where our optimization algorithms can bring us from above-breakeven to below-breakeven. We note that $n = 20$ and $r = 4$ for which the search space is already too vast for exhaustive search, with more than 10^{48} possible verification sequences. Indeed, even within an optimization subgroup $\langle V \rangle$ of rank $r = 4$, a total of 50,625 possible non-trivial verification sequences is far greater than the 500 $p_{\log}$ evaluations we used and proves challenging in practice for exhaustive search.

7 Experimental Demonstration

As a proof-of-concept intended to evaluate suitability of CliNR as a viable partial fault-tolerance noise-reduction technique in the short term, we also proceeded to implement several circuits on IonQ's Forte-Enterprise trapped-ion system. That system uses hyperfine states of a Ytterbium-171 ion as qubits, in a configuration with up to 36 qubits with all-to-all two-qubit gate connectivity.

We consider the CZNR version [9] of CliNR which implements a n-qubit Clifford circuit C, with $n = 10$, comprised purely of CZ gates using an resource state of n qubits (which is a graph state in this case) instead of $2n$ for the standard CliNR scheme.

We partition the 36 qubits of the machine into separate functional registers: (i) a 10-qubit input register R_{in}, onto which C is to be applied; (ii) two 10-qubit resource state registers $R^{(1)}_{\text{res}}$ and $R^{(2)}_{\text{res}}$, where the relevant resource states are prepared; (iii) a 6-qubit verification register R_{verif} used to measure the verification sequence.

Instead of an immediate restart upon failure of a

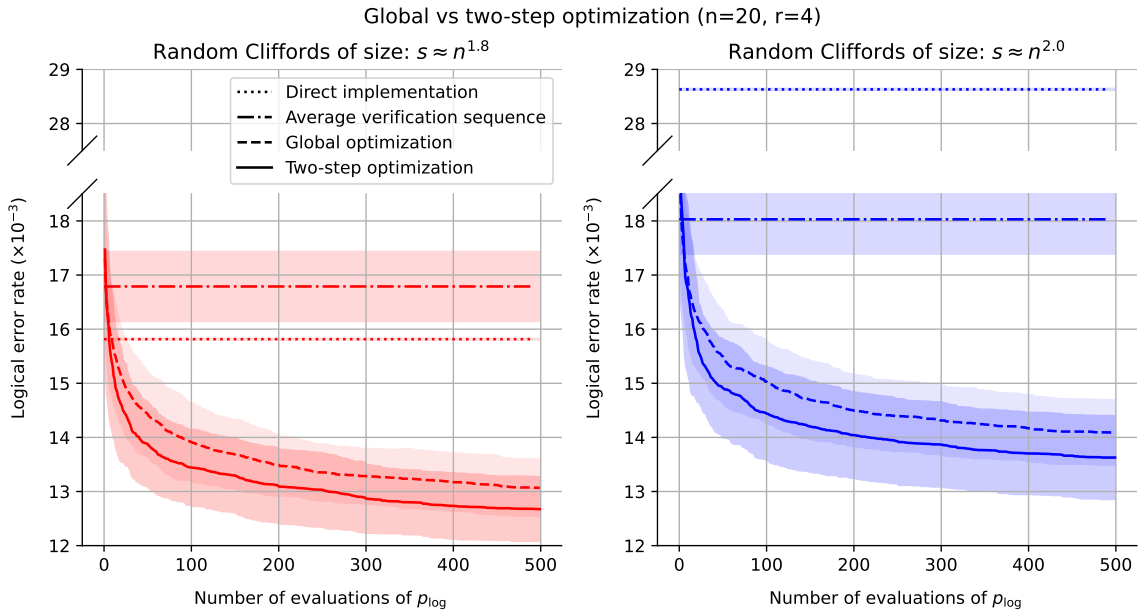

Figure 2: Comparison of the direct implementation, the CliNR implementation with a uniform random verification sequence proposed in [9] and the CliNR implementation with verification sequences optimized with global optimization (Algorithm 1) and two-step optimization (Algorithm 3). The standard error in logical error rate is shown in the shaded region around each curve.

verification sequence as in the standard description of CliNR and CZNR, we execute the entire CZNR circuit to completion whereupon measurement of R_{verif} will indicate whether a restart would have been necessary – the overall CZNR error rate is inferred by discarding such instances. Such an implementation obviates mid-circuit measurement and hardware reset of qubits.

Define the 10-qubit circuit

$$C = \prod_{1 \le i < j \le 10} CZ_{i,j} \tag{11}$$

which applies a CZ gate to every pair of qubits (45 in total). The experiment executes the circuit C^2 using CZNR with a random product stabilizer state as an input. It proceeds thus (we direct interested readers to Appendix B for further details):

1. C_{prep}, a product of n Haar-random single-qubit Clifford gates is chosen. The input state, $C_{\text{prep}}|0\rangle$ is prepared on R_{in}.

2. The resource state, defined as $C|+\rangle^{\otimes 10} = CH^{\otimes 10}|0\rangle^{\otimes 10}$, is prepared on $R^{(1)}_{\text{res}}$ and again on $R^{(2)}_{\text{res}}$.

3. Verification sequences are appended to $R^{(1)}_{\text{res}}$ and $R^{(2)}_{\text{res}}$, with qubits from R_{verif} being used for readout.

4. The circuit C is applied twice to R_{in} by consuming the resource states in $R^{(1)}_{\text{res}}$ and $R^{(2)}_{\text{res}}$. After this, the state $C^2 C_{\text{prep}}|0\rangle$ resides on $R^{(2)}_{\text{res}}$.

5. $C^{\dagger}_{\text{prep}}$ is applied to $R^{(2)}_{\text{res}}$.

6. All qubits are measured in the Z basis.

The fact that a CZ circuit is involutive means that the procedure above performs $C^2 = I$, the identity, on the state $C_{\text{prep}}|0^n\rangle$. In an ideal machine, the expected output after Step 6 is the all zero bitstring $|0^n\rangle$.

We ran each circuit in our experiment over 2,048 shots and retained per-shot data for analyses. Since our experiments do not rely on mid-circuit measurements and feedforward, restarts are accounted for in post-processing (shots with non-trivial stabilizer measurements are discarded, with the discarded fraction being used to infer restart rates). Similarly, post-teleportation Pauli corrections are realized as a bit-flips in measured bit-strings in post-processing. (See Appendix B for details).

Fig. 3 shows comparisons of the logical error rates between direct versus CliNR implementations of C. We note that CZNR circuits exhibit comparable logical error rates to circuits under direct implementation, even though the former contains up to $2n(r+1)$ additional two-qubit gates (e.g. 80 additional two-qubit gates, or $\approx 90\%$ more, for $r = 3$) than the direct implementation. We also note that even as r, the size of the verification sequence, increases (with a corresponding increase in

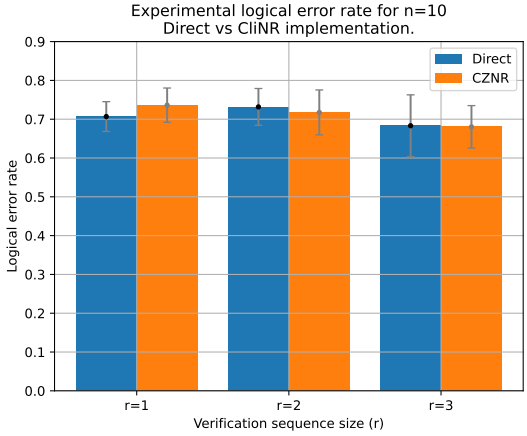

Figure 3: Experimental logical error rate comparing direct implementation (blue) with CZ noise-reduction or CZNR (orange) circuits implementating the same computation. In the three different experiments, verification sequence size (r) is increased, with no significant change in the logical error rate.

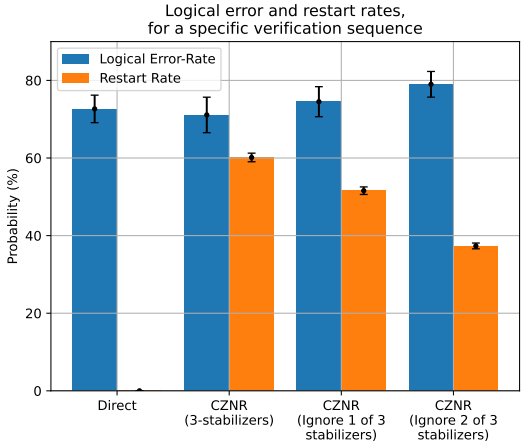

Figure 4: Comparison of the logical error rate of the CliNR implementations of the same Clifford circuit using a subset of the verification sequence. Restart rates shown here is inferred although the experiment is performed without mid-circuit measurement or restart.

gate count in the CZNR implementation), the logical error rate remains constant. A fraction of the noise is detected, manifesting as increased restart rate.

Further, to rule out noise-saturation – wherein all circuits are maximally noisy – as a reason for logical error rates remaining flat across implementations with different r, we implemented a circuit with $r = 3$, and then computed the logical error rate by progressively ignoring some outcomes of the verification sequence. If the measured stabilizers have any efficacy in detecting faults, then progressively neglecting them should increase logical error rates. Indeed, in Fig. 4 we observe exactly this – with the removal of each stabilizer in the verification sequence, the logical error rate worsens while the rejection/restart rate improves.

We *stress* that these results reflect circuit and device performance ***as is*** with no additional post-processing, filtering, or error-mitigation applied. Together, these provide evidence that CliNR as implemented on current IonQ's Forte Enterprise is at a breakeven noise threshold.

In these experiments, we note that the optimization techniques for verification sequences we described were not applied. To make these techniques applicable, one must first refine our cost function to integrate a more precise noise model reflective of hardware, for which previous benchmarking results could provide relevant insights [6].

8 Conclusion

In this work, we designed optimization algorithms to improve the performance of CliNR and its CZNR variant [9]. We designed algorithms to explore the search space more efficiently than naive random search and we establish a connection with the Grassmann manifold which allows us to significantly reduce the size of the search space. We design an optimization algorithm based on tabu search and, through numerical simulations, illustrate the potential gains in CliNR performance for the ion chain model. Finally, as a proof-of-concept illustrating the potential of CliNR to improve circuit performance in the near term, we also executed CliNR circuits on actual hardware and showed that even on current generation devices, CliNR is already at breakeven performance compared to direct circuit implementation.

Though we applied our ideas here to CliNR specifically, we note that exploiting symmetries of a search space within a stabilizer subgroup can be more widely applicable. Given that the stabilizer formalism is ubiquitous in quantum information and quantum error correction, we believe that our approach may find other applications in quantum computing. For example, it could be used to optimize stabilizer measurement sequences in quantum error correction as in [8, 23].

9 Acknowledgments

The authors thanks John Gamble for his feedback on a preliminary version of this work.

References

[1] Sara Bartolucci, Patrick Birchall, Hector Bombin, Hugo Cable, Chris Dawson, Mercedes Gimeno-Segovia, Eric Johnston, Konrad Kieling, Naomi Nickerson, Mihir Pant, et al. Fusion-based quantum computation. *Nature Communications*, 14(1): 912, 2023. DOI: https://doi.org/10.1038/s41467-023-36493-1.

[2] Thomas Bendokat, Ralf Zimmermann, and P-A Absil. A grassmann manifold handbook: Basic geometry and computational aspects. *Advances in Computational Mathematics*, 50(1):6, 2024. DOI: https://doi.org/10.1007/s10444-023-10090-8.

[3] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A—Atomic, Molecular, and Optical Physics*, 71(2):022316, 2005. DOI: http://doi.org/10.1103/PhysRevA.71.022316.

[4] Sergey Bravyi and Dmitri Maslov. Hadamard-free circuits expose the structure of the clifford group. *IEEE Transactions on Information Theory*, 67(7):4546–4563, 2021. DOI: http://doi.org/10.1109/TIT.2021.3081415.

[5] Hans J Briegel, David E Browne, Wolfgang Dür, Robert Raussendorf, and Maarten Van den Nest. Measurement-based quantum computation. *Nature Physics*, 5(1):19–26, 2009. DOI: https://doi.org/10.1038/nphys1157.

[6] Jwo-Sy Chen, Erik Nielsen, Matthew Ebert, Volkan Inlek, Kenneth Wright, Vandiver Chaplin, Andrii Maksymov, Eduardo Páez, Amrit Poudel, Peter Maunz, et al. Benchmarking a trapped-ion quantum computer with 30 qubits. *Quantum*, 8:1516, 2024. DOI: https://doi.org/10.22331/q-2024-11-07-1516.

[7] Dripto M Debroy and Kenneth R Brown. Extended flag gadgets for low-overhead circuit verification. *Physical Review A*, 102(5):052409, 2020. DOI: http://doi.org/10.1103/PhysRevA.102.052409.

[8] Nicolas Delfosse and Ben W Reichardt. Short shor-style syndrome sequences. *arXiv preprint arXiv:2008.05051*, 2020. DOI: https://doi.org/10.48550/arXiv.2008.05051.

[9] Nicolas Delfosse and Edwin Tham. Low-cost noise reduction for clifford circuits. *arXiv preprint arXiv:2407.06583*, 2024. DOI: http://doi.org/10.1103/PhysRevLett.134.090603.

[10] Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, 2021. DOI: https://doi.org/10.22331/q-2021-07-06-497.

[11] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986. DOI: https://doi.org/10.1016/0305-0548(86)90048-1. Applications of Integer Programming.

[12] Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989. DOI: https://doi.org/10.1287/ijoc.1.3.190.

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

[14] Daniel Gottesman. The Heisenberg representation of quantum computers. *arXiv preprint quant-ph/9807006*, 1998. DOI: https://doi.org/10.48550/arXiv.quant-ph/9807006.

[15] Daniel Gottesman and Isaac L. Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, 402(6760):390–393, November 1999. ISSN 1476-4687. DOI: https://doi.org/10.1038/46503.

[16] Emanuel Knill. Quantum computing with realistically noisy devices. *Nature*, 434(7029):39–44, 2005. DOI: https://doi.org/10.1038/nature03350.

[17] Simon Martiel and Ali Javadi-Abhari. Low-overhead error detection with spacetime codes. *arXiv preprint arXiv:2504.15725*, 2025. DOI: https://doi.org/10.48550/arXiv.2504.15725.

[18] John Preskill. Beyond NISQ: The Megaquop Machine. *ACM Transactions on Quantum Computing*, 6(3), April 2025. DOI: https://doi.org/10.1145/3723153.

[19] Robert Raussendorf and Hans J Briegel. A one-way quantum computer. *Physical review letters*, 86(22):5188, 2001. DOI: https://doi.org/10.1103/PhysRevLett.86.5188.

[20] Joschka Roffe, David Headley, Nicholas Chancellor, Dominic Horsman, and Viv Kendon. Protecting quantum memories using coherent parity check codes. *Quantum Science and Technology*, 3(3): 035010, 2018. DOI: http://doi.org/10.1088/2058-9565/aac64e.

[21] Peter W Shor. Fault-tolerant quantum computation. In *Proceedings of 37th conference on foundations of computer science*, pages 56–65. IEEE, 1996. DOI: https://doi.org/10.1109/SFCS.1996.548464.

[22] Andrew M Steane. Active stabilization, quantum computation, and quantum state synthesis. *Physical Review Letters*, 78(11):2252, 1997. DOI: https://doi.org/10.1103/PhysRevLett.78.2252.

[23] Theerapat Tansuwannont, Balint Pato, and Kenneth R Brown. Adaptive syndrome measurements for shor-style error correction. *Quantum*, 7:1075,

2023. DOI: https://doi.org/10.22331/q-2023-08-08-1075.

[24] Ewout van den Berg, Sergey Bravyi, Jay M Gambetta, Petar Jurcevic, Dmitri Maslov, and Kristan Temme. Single-shot error mitigation by coherent pauli checks. *Physical Review Research*, 5(3):033193, 2023. DOI: https://doi.org/10.1103/PhysRevResearch.5.033193.

[25] Min Ye and Nicolas Delfosse. Quantum error correction for long chains of trapped ions. *arXiv preprint arXiv:2503.22071*, 2025. DOI: https://doi.org/10.48550/arXiv.2503.22071.

A Proof of Proposition 1

Proof. Denote $\mathcal{V}$ the vertex set of $\mathrm{Gs}(S,r)$ and consider the restriction of α to the set $GL_r(\mathbb{Z}_2)\times\mathcal{V}$. Given that M is invertible, if $V\in\mathcal{V}$ then $V'=\alpha(M,V)$ is also in $\mathcal{V}$. To prove that this maps defined a group action, we need to show that $\alpha(I,V)=V$ and $\alpha(M,\alpha(N,V))=\alpha(MN,V)$. To prove this result, it is convenient to treat V as a $r\times 2n$ binary matrix that we denote $\bar{V}$. Then, the map α can be written as the matrix multiplication $\alpha(M,V)=MV$. From this description of α, the two group properties are immediately clear.

To prove that the proxy is invariant under the group action, it suffices to show that the condition $P\in V^\perp\backslash S^\perp$ and $P\in V'^\perp\backslash S^\perp$ are equivalent. If $P\in V^\perp$, then from Eq. (3) we have

$$[P,V_i']=\sum_{j=1}^{r}a_{i,j}[P,V_j]=0 \tag{12}$$

which proves that $P\in V'^\perp$. Therefore, we have $V^\perp\subseteq V'^\perp$. Moreover, we know that V and V' both generate rank-r subgroups, hence $V^\perp$ and $V'\perp$ are subgroups with the same rank $m-r$. Together with $V^\perp\subseteq V'^\perp$, this proves that $V^\perp=V'^\perp$ and therefore the cost function is invariant under the group action.

Consider the map $\sigma:\mathrm{Gs}(S,r)\to\mathrm{Gr}(S,r)$ which maps a vertex V of $\mathrm{Gs}(S,r)$ onto the vertex $\langle V\rangle$ of $\mathrm{Gr}(S,r)$. The map σ and the quotient map $\pi:\mathrm{Gs}(S,r)\to\mathrm{Gs}(S,r)/GL_r(\mathbb{Z}_2)$ form the following diagram.

$$\begin{array}{ccc}\mathrm{Gs}(S,r) & \xrightarrow{\ \sigma\ } & \mathrm{Gr}(S,r)\\ {\scriptstyle\pi}\downarrow & \nearrow_{\bar{\sigma}} & \\ \mathrm{Gs}(S,r)/GL_r(\mathbb{Z}_2) & & \end{array} \tag{13}$$

Therein, we use the notation $\mathrm{Gs}(S,r)$ and $\mathrm{Gr}(S,r)$ to represent the vertex sets of these two graphs. The existence of the map $\bar{\sigma}$ is guaranteed by the fact that σ is invariant under the action of the group $GL_r(\mathbb{Z}_2)$. Moreover, σ is surjective as any rank-r group admits admits at least one generating set with r elements. Therefore, $\bar{\sigma}$ inherits from the surjectivity of σ.

To show that $\bar{\sigma}$ is injective, consider two vertices V and V' of the search graph. If $\bar{\sigma}(V)=\bar{\sigma}(V')$ then V and V' generate the same subgroup of S. In particular, each element V_j' of V' can be decomposed into the elements of V. Denote

$$V_j'=\prod_{k=1}^{r}V_k^{m_{j,k}} \tag{14}$$

this decomposition. Similarly, let

$$V_i=\prod_{j=1}^{r}V_j'^{n_{i,j}} \tag{15}$$

be the decomposition of V_i into elements of V'. These decompositions define two $r\times r$ binary matrix $M=(m_{i,j})$ and $N=(n_{i,j})$. To prove that V and V' are in the same orbit under the action of $GL_r(\mathbb{Z}_2)$, it suffices to show that M is invertible because Eq. (14) can be written as $V'=\alpha(M,V)$. Plugging Eq. (14) inside Eq. (15), we get

$$V_i=\prod_{j=1}^{r}\left(\prod_{k=1}^{r}V_k^{m_{j,k}}\right)^{n_{i,j}}=\prod_{k=1}^{r}V_k^{\sum_{j=1}^{r}n_{i,j}m_{j,k}} \tag{16}$$

which holds for all $i=1,\dots,r$. Therein, we use the fact that the V_k commute which due to the fact that they belong to a stabilizer group S. From Eq. (16), we obtain

$$\sum_{j=1}^{r}n_{i,j}m_{j,k}=\delta_{i,k} \tag{17}$$

for all i,k, because the operators V_i are independent. In matrix terms, this reads $MN=I$, proving that M and N belong to $GL_r(\mathbb{Z}_2)$. Therefore V and V' are equal in the quotient $\mathrm{Gs}(S,r)/GL_r(\mathbb{Z}_2)$. This conclude the proof of the injectivity of $\bar{\sigma}$ and which proves that this map is a bijection between $\mathrm{Gs}(S,r)/GL_r(\mathbb{Z}_2)$ and $\mathrm{Gr}(S,r)$. □

B Experimental Details

B.1 Experimental Circuit and Verification Sequences

The Clifford circuit C we chose to implement is defined as:

$$C=\prod_{i<j}CZ_{i,j}$$

with $1 \leq i, j \leq n$. When applied to the state $|+^n\rangle$, it prepares the fully connected graph state.

In the absence of verification sequence optimization that is hardware-adapted to our machine's noise, we chose lowest- (highest-) weight stabilizers of $C|+\rangle^{\otimes n}$. These represent verification sequences that require the least (most) additional two-qubit gates to implement respectively.

Low-weight stabilizers have the form $Y_i Y_j$ with $i \neq j$ and $1 \leq i, j \leq n$. When $r > 1$, several such low-weight stabilizers are chosen at random with non-overlapping i and j.

By contrast, high-weight stabilizers have the form $X_i \prod_{j \neq i}^{n} Z_j$, with $1 \leq i, j \leq n$. When $r > 1$, several such high-weight stabilizers are chosen at random with non-overlapping i.

For both low- and high-weight stabilizers, we perform the experiment with a verification sequence with size $r = 1, 2, 3$.

B.2 Compilation and Gate Counts

All circuits are first compiled into the IonQ native gateset. That gateset is comprised of G_π, $G_{\pi/2}$, and ZZ gates, defined in terms of their action on the Z eigenbasis as follows:

$$G_\pi^\phi = \begin{bmatrix} 0 & e^{-i\phi} \\ e^{i\phi} & 0 \end{bmatrix}$$

$$G_{\pi/2}^\phi = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -ie^{-i\phi} \\ -ie^{i\phi} & 1 \end{bmatrix}$$

$$ZZ_\phi = \begin{bmatrix} e^{-i\phi/2} & & & \\ & e^{i\phi/2} & & \\ & & e^{i\phi/2} & \\ & & & e^{-i\phi/2} \end{bmatrix}$$

In that compilation process ***no*** optimization passes were applied to circuits in either CZNR or direct implementations; this ensures a fair comparison independent of compilation technique.

Post-compilation, the chosen C (comprised of a block of 45 CZ gates) reduces to 45 $ZZ_{\pi/4}$ gates, along with a layer of "Virtual-Z" operators that are not implemented.

The layer of Haar-random Clifford single-qubit gates C_{prep} that sets the input state and measurement bases is comprised of exactly one layer of n $G_{\pi/2}$ or n G_π gates. It is defined as $C_{\text{prep}} = \prod_{q=1}^{n} \left(G_\pi^{(0)}\right)^{c_q \cdot a_q} \left(G_{\pi/2}^{(a_q + b_q/2)\pi}\right)^{\neg c_q}$. That is, for each qubit q we choose a set of random bits $a_q, b_q, c_q \in \{0, 1\}$; then, if a_q and c_q are both '1', we apply $G_\pi^{(0)}$; otherwise, if $c_q = 1$, we compute $\phi_q = (a_q + b_q/2)\pi$ and then apply $G_{\pi/2}^{\phi_q}$ to that qubit.

The layer of $H^{\otimes n}$ at the start of CliNR stabilizer resource state preparation on $R_{\text{res}}^{(1)}$ and $R_{\text{res}}^{(2)}$ each adds a layer of n $G_{\pi/2}^{\pi/2}$ gates.

Implementing each stabilizer of weight w in the verification sequence, introduces an additional w $ZZ_{\pi/4}$ gates, along with up to two $2w$ G_π and $2w$ $G_{\pi/2}$ gates.

Finally, each CliNR teleportation block introduces an additional n $ZZ_{\pi/4}$ gates, along with $2n$ $G_{\pi/2}$ gates.

B.3 Post-Processing

In direct implementation of C, the logical error rate as shown in Figs. 3 and 4 are computed simply as $p_{\log} = 1 - \text{Prob}(0^n)$, the latter being the probability of the bitstring "000...".

In CZNR implementations, bits arising from stabilizer measurements are first inspected. All instances where one or more bits from measuring R_{verif} take value '1' are marked as "failed" (suppose there are N_{fail} of these). The restart rate is *inferred* as: $p_{\text{restart}} = \frac{N_{\text{fail}}}{N_{\text{shots}}}$, with $N_{\text{shots}} = 2048$.

Next, we inspect only instances **not** marked as "failed". For each instance, denote with $\vec{b}_0$, $\vec{b}_1$, $\vec{b}_2$ the bitstring obtained from measuring the registers R_{in}, $R_{\text{res}}^{(1)}$, and $R_{\text{res}}^{(2)}$. We then compute a correction Pauli P_{corr}:

$$\begin{aligned} P_{\text{corr}} &= C_{\text{prep}} C X^{\vec{b}_1 \oplus \vec{b}_0} C C_{\text{prep}}^\dagger \\ &= X^{\vec{x}} Z^{\vec{z}} \end{aligned}$$

Finally, we obtain the corrected output bitstring as: $\vec{b}_{\text{CZNR}} = \vec{b}_2 \oplus \vec{x}$. Now suppose there are N_0 instances in which $\vec{b}_{\text{CZNR}} = 0^n$. We *infer* the logical error rate as:

$$p_{\log} = \frac{N_0}{1 - N_{\text{fail}}}$$

B.4 Restart Rates

In Fig. 5 we show inferred restart rates as computed in Appendix B.3.

When $r > 1$, high-weight stabilizers in the verification sequence can lead to higher restart rates than their low-weight counterparts. Especially, when there are multiple high-weight stabilizer measurements, noise in earlier stabilizer measurements can lead to nontrivial outcomes in later ones, resulting in higher restart rates without corresponding improvement in logical error rates. We observe this in simulations, and in experimental data. Here, high-weight stabilizer measurements comprise up to $\approx 27\%$ and $\approx 35\%$ of total two-qubit gates for $r = 2$ and $r = 3$ respectively).

We also note that, these restart rates reflect the rate at which *any* stabilizer measurement of ancillae in both

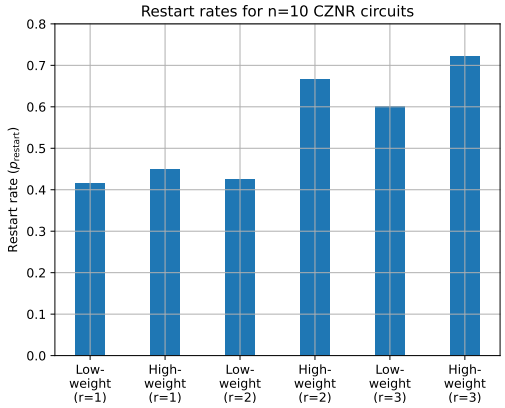

Figure 5: Inferred restart rates (see Appendix B.3) for CZNR circuits.

$R_{\text{res}}^{(1)}$ and $R_{\text{res}}^{(2)}$ is non-trivial. A comparable implementation of CliNR in the presence of mid-circuit measurement and restart will yield much higher circuit throughput, with an effective restart rate approximately given by $1 - \sqrt{1 - p_{\text{restart}}}$ (where p_{restart} is the restart rate shown here for this experiment). That distinction lies at the heart of what sets CliNR apart from techniques like the coherent parity check (CPC) [7, 17, 20, 24].