DQCode: Fault-Tolerant Zero State Preparation for Small qLDPC Codes on Near-Term Distributed Quantum Architectures

by Tim Neumann

Student Number: 6138721
Defence Date: 25 August 2026

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Quantum Information Science and Technology

Graduation committee

Dr. Sebastian Feld	Chair
Dr. Tim Coopmans	Committee Member
Dr. Maximilian Rimbach-Russ	Committee Member
Luise Prielinger, MSc. MSc.	Committee Member

An electronic version of this thesis is available at https://repository.tudelft.nl/.

Copyright © 2026 Tim Neumann. Licensed under CC BY 4.0.

Plan the work, work the plan.

Abstract/Samenvatting

The intersection of Quantum Error Correction (QEC) and Distributed Quantum Computing (DQC) promises a pathway to scalable fault-tolerant quantum computing; in particular, qLDPC codes and Type-II architectures both constitute candidates for near-term implementations. In this work, we introduce `DQCode` for optimised fault-tolerant logical $|0\rangle_L^{\otimes k}$ state preparation of the Trivariate Bicycle-$[[12,2,3]]$ and Bivariate Bicycle-$[[18,4,4]]$ qLDPC codes on distributed Type-II quantum systems.

Based on a QEC cycle-aware qubit mapping, we use a Genetic Algorithm to discover state preparation circuits with optimised telegate counts. Combined with verification circuits for post-selection of runs, this enables efficient fault-tolerant state encoding, which we simulate with open-source `Julia` software in a realistic DQC setting under circuit-level and Bell pair initialisation noise. Across codes and distributed architectures, we obtain approximately quadratic scaling of the logical initialisation error rate (LIER) with respect to the physical error rate p under suitable conditions, matching theoretical expectation. Furthermore, we find pseudothresholds of $p_{\text{th}} \approx 10^{-4}$ and $p_{\text{th}} \approx 10^{-5}$ for the Trivariate and Bivariate Bicycle codes, respectively, below which the LIER outperforms p for sufficiently high remote-entanglement fidelities. Despite noise floors for very small p, all systems tolerate remote entanglement being noisier than local operations at the identified thresholds, while maintaining high post-selection acceptance rates.

Assuming improvements of DQC candidate platforms, our results indicate the near-term viability of encoding small qLDPC codes on Type-II architectures. By providing circuits for proof-of-principle experiments of small-scale instances, we expect our work to benefit the benchmarking of early near-term distributed systems.

Summary of Contributions & Impact

In this work, we discover novel DQC-tailored fault-tolerant state preparation circuits for small Multivariate Bicycle codes as well as other CSS codes, which can directly be used for proof-of-concept experimental demonstration of logical encoding on Type-II distributed quantum hardware. To this end, `DQCode` (i) extends existing state preparation research to distributed Type-II systems while maintaining fault tolerance of the procedure, and (ii) widens existing research of Type-II systems towards qLDPC codes. We obtain numerical results suggesting the feasibility of encoding logical $|0\rangle_L^{\otimes k}$ states for qLDPC codes on near-term distributed machines, such as ion traps with photonic interconnects. Even though the code sizes we investigate here allow for implementation on many state-of-the-art monolithic machines, early near-term distributed systems will likely benefit from such small-scale instances for testing and benchmarking. Furthermore, our ideas generalise to larger codes, serving as a stepping stone for using qLDPC codes on Type-II devices.

From a software perspective, we expose `DQCode` as an open-source `Julia` tool. This contributes a DQC adaptation of the overlap compilation method and a Genetic Algorithm as optimisation tools, as well as a DQC Clifford simulator for specified networking assumptions and noise conditions. On top of producing the results presented herein, this tool enables researchers to optimise and assess the encoding of QEC codes for custom noise models on user-specified architectures. As such, the `DQCode` pipeline allows for automated investigation of distributed encoding circuits for small CSS codes.

Statement on AI usage

The author acknowledges the use of Artificial Intelligence (AI). In particular, *Claude Sonnet* 5 was used for tasks including brainstorming, sanity checking and graphical formatting of text, while *GitHub Copilot* (with varying underlying models) provided coding support for plot design and occasional implementations of pseudocode. No agentic AI was used in the work. The author takes full responsibility of all code and written output of this thesis.

Interdisciplinarity of `DQCode`

This work is settled at the intersection of (i) Quantum Error Correction and Distributed Quantum Computing in terms of quantum research fields, and (ii) Quantum Computing and Quantum Communication in terms of quantum technologies. As such, `DQCode` draws inspiration from different disciplines, closely aligned with the interdisciplinary character of the study program. Furthermore, the project is inspired by software-hardware co-design in the form of investigating qLDPC codes on Type-II distributed architectures as particular code-architecture pair.

Acknowledgements

This thesis is the result of two intense years in Holland, filled with the intellectual joy of breaking into a new and fascinating area of knowledge, but also not devoid of times of overwhelming challenge. My journey would not have been possible, and in any case not as fulfilling, without the people I was fortunate to already have in my life, as well as the people I got to meet throughout my time in Delft; as friends, colleagues, supervisors, teachers, mentors and teammates.

To my thesis supervisors Sebastian and Luise, who not only helped shape this project with sharp feedback and great patience, but also granted me the freedom to explore my interests. "Nobody cares how much you know until they know how much you care" — Ich danke euch, dass ich mich zu jeder Zeit auf euch verlassen konnte.

To Tim Coopmans and Maximilian Rimbach-Russ, for being part of my thesis committee and making time in their busy schedules to read my work.

To the quantum research community, for being supportive and open. In particular, many thanks to Stefan Krastanov, Tom Peham, Siddhant Singh, Sébastian de Bone, Evan Sutcliffe, Remmy Zen, Eliška Greplová, Christian Andersen, and Eric Kuehnke, with whom I had many fruitful discussions. Thank you also to Barbara Terhal, Feroz Mian and Naoyuki Kanomata for answering my questions.

To all the people that have shaped the quantum field, on whose shoulders we stand.

To the Feld Group, for being a testbed for ideas, and in particular to Mankrit, Akash and Noa for the discussions outside of group meetings.

To my fellow QIST students, forming a diverse community of students with wildly different backgrounds, to learn from and grow with one another.

To my professors, for treating students as equals. Thank you especially to Jos and Jordi for guiding me through my first TA experience.

To the QIST administration, whose hard work made this program possible in the first place.

To Miriam, for caring deeply about everyone and teaching me so many things about life. You're my reminder that being successful and being kind can go hand in hand.

To the people living in corridor 21 over the last two years, for providing a warm place to come home to every day.

To Daniela, Alice and Teresa, for the unexpected friendships we make along the way.

To my teammates in Ariston and the run club, for being a driven and warm-hearted community.

To César, Arturo, Andrea, Nikhil and Vittorio, for making Delft feel like home from the beginning. Discussing physics over dinner, doing hackathons together and sharing the wins and losses of everyday life — you were the best friends I could have wished for.

To my parents, for visiting me in Delft, even by bike, whenever I got too absorbed in work to make it home, for providing endless cake supplies and sharing way too many tapas. Ihr wart für mich da an meinen besten und schlechtesten Tagen, und ohne euch hätte ich es niemals bis hierher geschafft. Ich verdanke euch alles.

To my sister, for being a true inspiration in everything you do. We were far but never apart over the last six years, and to struggle, win, laugh and grow together with you meant the world to me. Marylin, Danke für all die Telefonate, in denen wir uns in den kleinen und großen Dingen des Lebens verloren haben, und dass du niemals müde wirst, un poco loco zu sein.

Tim Neumann
Delft, August 2026

Contents

1 Introduction **1**

2 Theoretical Background and Related Work **2**
2.1 Quantum Error Correction . . . 2
2.1.1 From Classical to Quantum Error Correction . . . 2
2.1.2 Stabiliser Codes: Stabiliser Group and Codespace . . . 3
2.1.3 Logical Pauli Operators . . . 5
2.1.4 Error Correction . . . 7
2.1.5 Clifford Formalism . . . 10
2.1.6 Canonical Form . . . 13
2.1.7 CSS Codes . . . 15
2.1.8 qLDPC Codes . . . 16
2.1.9 Universal Quantum Computing . . . 18
2.1.10 Fault Tolerance . . . 19
2.1.11 Logical State Encoding . . . 21
2.2 Distributed Quantum Computing . . . 26
2.2.1 Quantum Gate Teleportation and Circuit Cutting . . . 26
2.2.2 DQC Compilation . . . 28
2.2.3 DQC Hardware . . . 29
2.3 QEC meets DQC: Distributed Stabiliser Codes . . . 30

3 Project Outline **32**
3.1 Motivation . . . 32
3.2 Research Questions . . . 34

4 Optimised Synthesis of State Preparation Circuits for Type-II Architectures **36**
4.1 Optimisation Preliminaries . . . 36
4.1.1 CSS Code Encoding Circuit Synthesis . . . 36
4.1.2 Hardware Assumptions . . . 37
4.1.3 QEC-Aware Qubit Mapping via Hypergraph Partitioning . . . 38
4.1.4 Setting Up Fault Tolerance . . . 40
4.2 Optimisation Methods . . . 42
4.2.1 DQC-FT Compilation . . . 43
4.2.2 Warm-Start Genetic Algorithm . . . 44
4.2.3 Monte Carlo Tree Search . . . 49
4.2.4 Baseline Methods . . . 52
4.3 Results & Discussion . . . 54
4.4 Limitations & Future Work . . . 57

5 DQC Simulation of Fault-Tolerant Encoding Circuits **59**
5.1 Verification Circuits For Fault-Tolerant State Preparation . . . 59
5.2 Simulation Methods . . . 64
5.2.1 DQC Error Model . . . 64
5.2.2 Scheduling Circuit Operations . . . 67
5.2.3 Lookup Table Decoding . . . 71
5.2.4 Metrics . . . 71

5.2.5 DQC Simulation Design . . . 77
5.3 Results & Discussion . . . 80
5.4 Limitations & Future Work . . . 94

6 Conclusion & Outlook **96**

A `DQCode` Software **i**
A.1 Data and Code Availability . . . i
A.2 Reproducibility . . . i
A.3 `DQCode` Code Structure . . . i

B Supplementary Information and Derivations **iii**
B.1 Stabiliser code projector . . . iii
B.2 Stabiliser Measurement . . . iii
B.3 Threshold theorem . . . iv
B.4 Transversal CX gate between two Steane codes . . . iv
B.5 Derivation of the Canonical Logical Operators . . . v
B.6 Canonicalisation in `QuantumCliiford.jl` . . . viii
B.7 Codespace Dimension for CSS Codes . . . ix
B.8 Commutativity in CSS Code Tableau . . . ix
B.9 Gottesman Encoding Circuit . . . x
B.10 Telegate Protocol . . . xvi
B.11 Stabiliser Tableaus of Selected QEC Codes . . . xvii
B.12 DQC Compilation of Gottesman Encoding Circuit . . . xxi
B.13 Non-FT State Preparation Circuits for Selected QEC Codes . . . xxii
B.14 From Markov Decision Process to Monte Carlo Tree Search . . . xxvi
B.15 Monte Carlo Tree Search: Remarks . . . xxviii
B.16 State Preparation and Measurement Error . . . xxix
B.17 Idling Error Probability Per Telegate Round . . . xxx
B.18 LIER and Logical Z-measurements . . . xxx
B.19 Standard Deviation of $\overline{\text{LIER}_i}$ . . . xxxii
B.20 DQC Simulation: Mapping and Operations . . . xxxiii

1 Introduction

Quantum technologies bear great potential in many areas of science and business, promising large-scale optimisation of combinatorial problems [1], and posing new threats to cryptography [2], for example. The quantum bit, or *qubit*, is the quantum-mechanical analogue of a classical bit, acting as the building block of many such technologies [3]. Qubits are two-level systems, exhibiting a range of enticing properties, such as quantum superposition and entanglement; however, quantum systems are generally very susceptible to noise, and so far most technologies are restricted to a relatively small number of qubits [4, 5]. To achieve quantum utility, this necessitates the study of how to protect fragile quantum information (Quantum Error Correction) [6], and how to solve the scaling bottleneck of monolithic quantum chips, e.g., by transitioning to multi-core distributed systems (Distributed Quantum Computing) [7].

In this project, we work at the intersection of these two fields, investigating how to encode quantum information on a distributed quantum system, consisting of multiple Quantum Processing Units (QPUs) connected via a quantum network. Therefore, we introduce `DQCode` as a pipeline for (i) optimising quantum circuits that map a physical quantum state to its logical (i.e., protected) counterpart on a specific class of distributed quantum systems, and (ii) assessing the quality of this encoding routine numerically[1]. As such, `DQCode` is also located at the interdisciplinary intersection of quantum communication and computing, which (next to quantum sensing) constitute two of the three major fields in quantum technology.

In particular, we investigate preparing the $|0\rangle_L^{\otimes k}$ state for multiple state-of-the-art quantum error correction codes, paired with a realistic circuit-level noise model adapted to the distributed setting. The overall goal herein is to provide a fault-tolerant state preparation procedure that can directly be used for proof-of-principle demonstrations on near-term distributed quantum systems. Drawing inspiration from the `DQCode` pipeline, this thesis is structured in multiple sections. Firstly, in Section 2, we develop the necessary theory of Quantum Error Correction (QEC) and Distributed Quantum Computing (DQC), including a review of related work. This is followed by an outline of the precise research motivation, scope and objectives in Section 3. We then examine our optimisation work in Section 4, demonstrating how a Genetic Algorithm for distributed CX circuit synthesis effectively reduces communication overhead between QPUs when preparing a logical state. In Section 5, we assess the quality of these optimised circuits in a realistic DQC setting via numerical simulation, based on a number of encoding metrics defined throughout. Within each of these sections, we discuss and contextualise our results as they relate to our research questions, and provide avenues for future work. Finally, Section 6 summarises our main findings and places this work in a broader context of distributed fault-tolerant quantum computing.

[1] All code and data for this project is available open source, as desribed in Appendix A.1.

2 Theoretical Background and Related Work

In the following, we introduce Quantum Error Correction (QEC) and Distributed Quantum Computing (DQC), both of which have received major attention from the quantum research community in recent years. Our goal is not to provide a comprehensive review of these fields —[2] rather, we strive to give a synopsis of their key concepts as they relate to DQCode, providing the necessary background to introduce the distributed encoding of logical states. Therefore, in Section 2.1, we develop the theory of stabiliser codes, fault tolerance and logical state encoding, while in Section 2.2, we introduce central concepts of distributed systems and quantum networks. To contextualise the research landscape that DQCode contributes to, we point to influential literature shaping the respective domains throughout.

We assume familiarity with fundamental notions and notation in Quantum Information Theory as well as group theory, yet no specific prior knowledge in QEC or DQC. Given the respective prerequisite knowledge, the reader may wish to skip to Section 2.3, which treats error correction on distributed systems, and return back to definitions and concepts introduced in Sections 2.1 and 2.2 as needed.

2.1 Quantum Error Correction

Quantum Error Correction techniques have emerged as powerful tools to tackle error-prone qubits and quantum gates, and pave the path to large-scale fault-tolerant computing. One can find excellent treatments of this active field of research in literature [6, 3, 8, 9].

2.1.1 From Classical to Quantum Error Correction

Even though quantum information does not play by the same rules as classical information (e.g., it is impossible to copy a general quantum state [10]), QEC draws upon a plethora of classical error-correction techniques, which were developed starting in the mid-20$^{\text{th}}$ century with the advent of digital computers [11] and digital communication [12]. A central object in classical coding theory is a *(linear) code*, which is a (linear) subspace of the n-dimensional vector space $\mathbb{F}_q^n$, with $\mathbb{F}_q$ being the finite field of order q (such that $q = 2$ implies binary encoding). We refer to this subspace as the *codespace* C, and to its elements as *codewords*.

The intuition behind this construction is that low-weight bit errors, which could result from transmission of a codeword via a noisy channel, for example, can sometimes be corrected, since the overhead of projecting into a higher-dimensional space increases the relative distance between codewords. In essence, we thus use coding theory to trade-off the efficiency of representing information (e.g., a message) with our ability to correct undesired errors that this information might suffer from.

We can conveniently express C with a so-called *parity-check matrix* H, which imposes parity constraints on the codewords via the condition

$$Hc = \mathbf{0} \mod 2 \quad \forall c \in C.$$

This allows us to identify *error syndromes*, indicating the precise location of certain bit errors, as the below example illustrates.

[2] We would like to stress that all em-dashes in this thesis are humanly-generated.

Example 1: Error correction with the binary Hamming code

The Hamming-[7,4] code is captured by the parity-check matrix

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Any 7-dimensional binary vector c in the kernel

$$\ker(H) \equiv \{c \in \mathbb{F}_2^7 : Hc = \mathbf{0}_7 \mod 2\} \tag{1}$$

of H is a codeword of the Hamming code, such that we embed a $|\ker(H)| = 4$-dimensional binary subspace of $2^4 = 16$ codewords into a 7-dimensional ambient space. If a single bit error e_i occurs during transmission of the Hamming codeword $c = [1,0,0,1,1,0,0]^T$, for example, we can recover c by computing $H(c+e_i) = Hc + He_i = He_i$ on the incoming message $c+e_i$, and reversing the bit-flip on the i^{th} bit. In particular, He_i is the i^{th} column vector of H, and thus tells us the precise error location, whereas an error on a transmitted unencoded 4-bit word would have been irrevocable.

When constructing QEC codes, we leverage a similar principle, adding redundancy to the information in order to make it more robust against errors. We will shortly see how the quantum versions of parity checks, code distance and error syndromes enable the protection of quantum information.

2.1.2 Stabiliser Codes: Stabiliser Group and Codespace

Over the past decades, various quantum codes have been introduced. The very first quantum code constructions by Shor [13] and Steane [14] were soon subsumed under the notion of *CSS* codes (cf. Section 2.1.7) in the 1990s [15, 16]. These pioneering works were then generalised in the framework of the most prominent class of quantum codes, the so-called *stabiliser codes*, first comprehensively introduced by Gottesman in his dissertation in 1997 [17], and independently developed by Calderbank, Rains, Shor and Sloane [18]. A schematic overview of stabiliser codes is given in Figure 1. The goal of this subsection is to introduce the components of the graphic and introduce the relevance of defining a logical qubit space (represented by the blue cube in Figure 1).

Stabiliser codes are defined via so-called *stabilisers*, which are strings (tensor products) of the standard Pauli matrices

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \text{ and } Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2}$$

A set of $n-k$ independent and commuting stabilisers $\{S_i\}_{i=1}^{n-k}$ with $S_i \neq \mathbb{1}_{\otimes n}$ and $0 \leq k \leq n$[3] generate the abelian stabiliser subgroup $\mathcal{S} = \langle S_1, \ldots, S_{n-k} \rangle$ of the n-qubit Pauli group $\mathcal{P}_n$,

$$\mathcal{P}_n = \left\{ c \cdot s_1 \otimes \ldots \otimes s_n : c \in \{\pm 1, \pm i\}, s_i \in \{\mathbb{1}, X, Y, Z\} \right\}. \tag{3}$$

[3] Note that the index i for a stabiliser generator S_i runs over the number of generators, and *not* over the number of qubits, i.e., S_i is a Pauli string potentially acting on all n qubits.

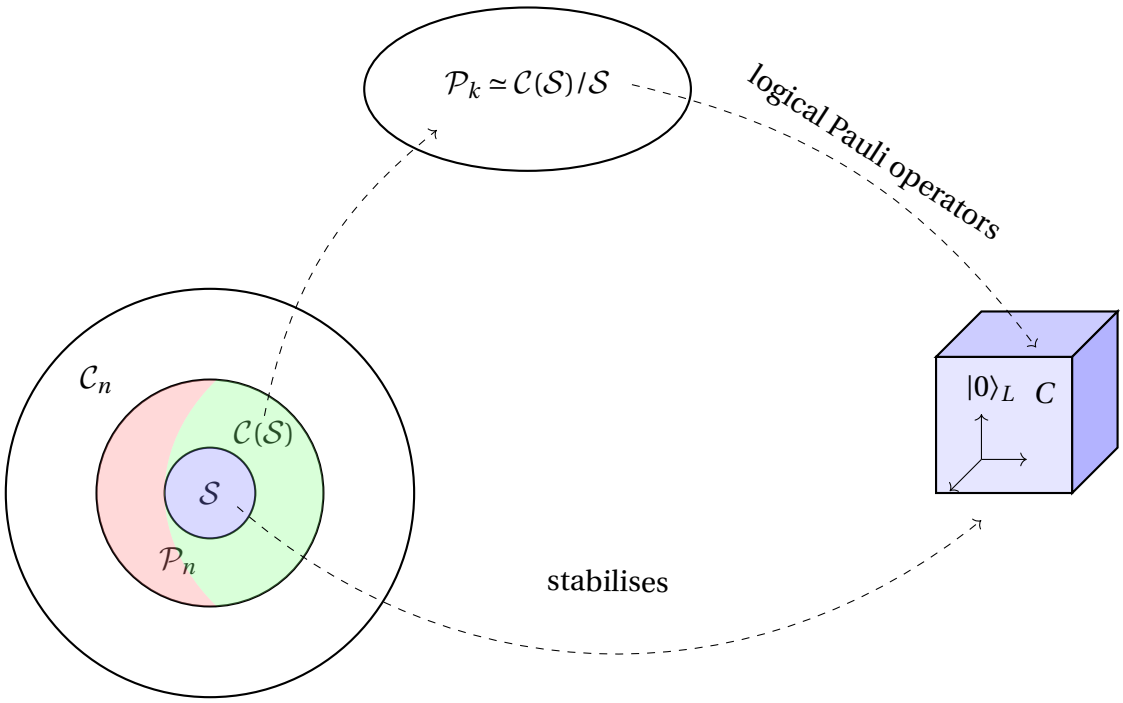


Figure 1: Diagram connecting central notions of stabiliser codes, including stabiliser subgroup $\mathcal{S}$, Pauli group $\mathcal{P}_n$, Clifford group $\mathcal{C}_n$, logical Pauli operators $\mathcal{P}_k$ and logical codespace C. The centraliser $\mathcal{C}(\mathcal{S})$ contains $\mathcal{S}$, and the red shade indicates detectable errors in $\mathcal{P}_n \setminus \mathcal{C}(\mathcal{S})$.

We require that $-\mathbb{1}_{\otimes n} \notin \mathcal{S}$, where $\mathbb{1}_{\otimes n} = \mathbb{1}_{2^n}$ is the identity operator on the n-qubit space. Then, the codewords of the stabiliser code C are precisely those quantum states $|\psi\rangle$ (in the 2^n-dimensional complex Hilbert space $\mathcal{H}$ for n qubits) that belong to the shared $+1$-eigenspace of $\mathcal{S}$, i.e.,

$$C = \left\{ |\psi\rangle \in (\mathbb{C}^2)^{\otimes n} : S|\psi\rangle = |\psi\rangle \; \forall \, S \in \mathcal{S} \right\}. \tag{4}$$

Since every stabiliser generator of $\mathcal{S}$ effectively halves the space of admissible quantum states by adding its $+1$-eigenspace as restriction, the codespace exhibits complex dimension $2^n/(2^{n-k}) = 2^k$. Building up this intuition, a set of $n-1$ stabilisers defines a $2^{n-(n-1)} = 2$-dimensional codespace — a *qubit* — whereas a full set of n stabilisers defines a $2^{n-n} = 1$-dimensional codespace – a single quantum state up to global phase[4].

The labeling with n (for the ambient space) and k (for the encoded space) here directly relates to classical coding theory, where $[n,k]$-codes embed a k-dimensional codespace in an n-dimensional ambient space. In that sense, the $n-k$ stabilisers above play a similar role to the $n-k$ rows of a classical parity-check matrix in Section 2.1.1, which define the classical codespace of dimension k as their kernel. This connection will become even more apparent in Section 2.1.7, where we introduce how quantum CSS codes are constructed from classical codes.

For quantum codes, we usually refer to the n actual qubits on some quantum processor (living in the 2^n-dimensional Hilbert space $\mathcal{H}$) as *physical qubits*, and to the k *logical qubits* as the encoded qubits in the 2^k-dimensional codespace. Again, the rationale behind this encoding is that we trade off efficiency for redundancy — sacrificing

[4] It would not be very useful to define a codespace containing a single quantum state, of course.

capacity in the larger Hilbert space, making the information in the codespace more robust against errors.

Example 2: Steane code stabilisers

The Steane code yields $k = 1$ logical qubits for seven physical qubits, defined by the six generators:

$$S_1 = XIXIXIX,\ S_2 = IXXIIXX,\ S_3 = IIIXXXX,$$
$$S_4 = ZIZZIZI,\ S_5 = ZZIIZZI,\ S_6 = ZZIZIIZ,$$

where the left-most Pauli operator acts on the first qubit, and each stabiliser is the tensor product of its constituent Paulis. This gives rise to the logical qubit states

$$\begin{aligned}|0\rangle_L = \frac{1}{2\sqrt{2}}\Big(&|0000000\rangle + |1010101\rangle + |0110011\rangle + |1100110\rangle \\ &+ |0001111\rangle + |1011010\rangle + |0111100\rangle + |1101001\rangle\Big),\\ |1\rangle_L = \frac{1}{2\sqrt{2}}\Big(&|1111111\rangle + |0101010\rangle + |1001100\rangle + |0011001\rangle \\ &+ |1110000\rangle + |0100101\rangle + |1000011\rangle + |0010110\rangle\Big).\end{aligned}$$

It will later become important that we can project a general quantum state onto the codespace. For this, we define the orthogonal projector Π_C as a function of the stabilisers,

$$\Pi_C = \prod_{i=1}^{n-k} \frac{\mathbb{1}_{\otimes n} + S_i}{2}. \tag{5}$$

Expression 5 is a result of all stabilisers being involutory Pauli strings, admitting eigenvalues ± 1 (cf. Appendix B.1).

The Pauli group holds special relevance for stabiliser codes, deriving from the action of Pauli strings on its subgroup $\mathcal{S}$. Next, we consider two different cases: (i) the part of $\mathcal{P}_n$ that commutes with $\mathcal{S}$ (which we will identify as the logical Pauli group in Section 2.1.3), and (ii) the part that anti-commutes, which hosts detectable errors, as discussed in Section 2.1.4. Any two distinct Pauli operators either commute or anti-commute, hence these two cases are comprehensive. This follows directly from the canonical (anti-)commutation relations

$$\sigma_i \sigma_j = \delta_{ij}\mathbb{1} + i\epsilon_{ijk}\sigma_k, \tag{6}$$

where σ is a placeholder for the Pauli operators, identifying $\sigma_1 \equiv X, \sigma_2 \equiv Y$ and $\sigma_3 \equiv Z$, and ϵ is the Levi-Civita symbol.

2.1.3 Logical Pauli Operators

Note that in order to perform operations on the encoded logical states in C, we require a set of logical operators, $\mathcal{L}$. The central requirement for a logical operator $L \in \mathcal{L}$ is that it maps logical states to logical states, i.e.,

$$L|\psi\rangle \in C\ \forall\ |\psi\rangle \in C. \tag{7}$$

As a special class among all such operators, let's consider a logical Pauli operator $L_{\mathcal{P}} \in \mathcal{P}_n$. Recall that $S|\psi\rangle = |\psi\rangle = S'|\psi\rangle$ holds for all $S, S' \in \mathcal{S}$, since $|\psi\rangle$ is a $+1$-eigenstate of all stabilisers in the stabiliser group. Also note that $\mathcal{S}$ is the unique subgroup of $\mathcal{P}_n$ stabilising C[5]. Then, for $L_{\mathcal{P}}|\psi\rangle = |\psi'\rangle$ we find

$$L_{\mathcal{P}} S|\psi\rangle = L_{\mathcal{P}}|\psi\rangle = |\psi'\rangle = S'|\psi'\rangle = S' L_{\mathcal{P}}|\psi\rangle \tag{8}$$

for all $S, S' \in \mathcal{S}$. Since this chain of equalities holds for all $|\psi\rangle \in C$, we have

$$L_{\mathcal{P}} S|_C = S' L_{\mathcal{P}}|_C. \tag{9}$$

Furthermore, any two Pauli operators either commute or anti-commute, so we have

$$L_{\mathcal{P}} S = \pm S L_{\mathcal{P}} \tag{10}$$

for a given stabiliser S. Together, Equations 9 and 10 yield $S' = \pm S$. Additionally, the requirement $-\mathbb{1}_{\otimes n} \notin \mathcal{S}$ ($-\mathbb{1}_{\otimes n}$ does not have a non-trivial $+1$-eigenspace) implies that $-S \notin \mathcal{S}$, and thus forces $S' = S$, showing that a logical Pauli operator commutes with any stabiliser $S \in \mathcal{S}$.

More abstractly, every logical Pauli operator lives in the centraliser $\mathcal{C}(\mathcal{S})$ of the stabiliser group, which contains all operators that commute with $\mathcal{S}$. In Figure 1, the centraliser is the union of the green region and $\mathcal{S}$, where $\mathcal{S}$ trivially commutes with itself. Herein, a non-trivial logical operator $L_{\mathcal{P}}$ should not act trivially on the codespace, hence it cannot be a product of stabilisers (which fix each element in C).

These conditions are captured in the mathematical structure of the quotient group $\mathcal{C}(\mathcal{S})/\mathcal{S}$. As such, logical Paulis are the equivalence classes, or cosets, of $\mathcal{C}(\mathcal{S})/\mathcal{S}$, which form the k-qubit *logical Pauli group*. Each equivalence class in $\mathcal{C}(\mathcal{S})/\mathcal{S}$ contains 2^{n-k} elements (each coset is closed under multiplication with the 2^{n-k} stabilisers), and since $\mathcal{C}(\mathcal{S})$ contains $4 \cdot 2^{n+k}$ elements [17, Ch. 3.2], there are exactly $4 \cdot 4^k$ such equivalence classes in the quotient group. The coefficient of 4 results from the four possible phases $\{\pm 1, \pm i\}$ for each Pauli string; ignoring these global phase factors, we find that the 4^k equivalence classes precisely match the requirement on the dimension of the Pauli group on a 2^k-dimensional codespace. This allows us to choose a set of logical operators $\{X_{L,i}, Y_{L,i}, Z_{L,i}\}_{i=1}^k$ admitting the canonical commutation relations from Equation 6 in logical operator space, and to identify them with the space $\mathcal{P}_k$ of Pauli strings of size k[6].

Choosing some coset representative as the logical operator becomes relevant when fixing a basis of our logical qubit space. In close analogy to the physical qubit space, we denote the logical computational basis states $|0\rangle_{L,i}$ and $|1\rangle_{L,i}$, such that by construction the state $|x_1 x_2 \dots x_k\rangle_L \equiv |x_1\rangle_{L,1} \otimes |x_2\rangle_{L,2} \otimes \dots |x_k\rangle_{L,k}$ is stabilised by

$$\left\langle \mathcal{S}, (-1)^{x_1} Z_{L,1}, (-1)^{x_2} Z_{L,2}, \dots, (-1)^{x_k} Z_{L,k} \right\rangle \qquad \text{(cf. [3, Ch. 10.5])} \tag{11}$$

for $x_i \in \{0, 1\}$.

[5] Assume to the contrary that this was not the case. Then there would exist at least one more element $s^* \in \mathcal{P}_n$ that stabilises C, in which case the elements in the set $\mathcal{S}^* = \mathcal{S} s^*$ would also stabilise C. Furthermore, $s^* \notin \mathcal{S}$ would imply that the number of independent generators of $\langle \mathcal{S}, s^* \rangle$ is strictly larger than that of $\mathcal{S}$, and that therefore the dimension of C is strictly smaller than itself, which is impossible.

[6] There exists a procedure to obtain the so-called *standard form* of stabiliser generators, will be discussed in Section 2.1.11. From this, we may obtain logical Pauli operators in a systematic way.

Example 3: Logical Pauli operators for the Steane code

From the standard form of the Steane code and following the procedure discussed in Section 2.1.11, we obtain $Z_L = Z_2 \otimes Z_4 \otimes Z_6$ and $X_L = X_3 \otimes X_5 \otimes X_6$ as logical operators. We may verify the expected relations

$$Z_L|0\rangle_L = |0\rangle_L, Z_L|1\rangle_L = -|1\rangle_L, \text{ and } X_L|0\rangle_L = |1\rangle_L. \tag{12}$$

Even though $\mathcal{L}$ contains many more logical operators than the logical Paulis, the study of $\mathcal{L}_{\mathcal{P}}$ lies at the heart of quantum error correction, as will become apparent in the following subsection.

2.1.4 Error Correction

> "Encoding a logical qubit redundantly into many physical qubits enables one to diagnose and correct errors by repeatedly measuring syndromes of parity check operators." – Sergey Bravyi, [19]

Apart from the logical Pauli operators that commute with $\mathcal{S}$, we next consider Pauli operators $E \in \mathcal{P}_n$ that anti-commute with at least one element in $S \in \mathcal{S}$, i.e.

$$ESE = -S \Leftrightarrow ES = -SE, \tag{13}$$

and belong to the red region in Figure 1. In contrast to Equation 9, where logical operators map stabilisers to stabilisers under conjugation, the operator E in Equation 13 maps a stabiliser to some non-stabiliser Pauli element $-S$. Such Pauli operators E are referred to as *detectable errors*, since the anti-commutation of E with some stabiliser S can be extracted by a *stabiliser measurement*. Such stabiliser measurements are projections of the current system state $|\psi\rangle$ onto the given stabiliser; when we speak of measuring a stabiliser S, mathematically we refer to projecting the system onto an eigenspace S. This projection is implemented via an indirect measurement, as shown Figure 2.

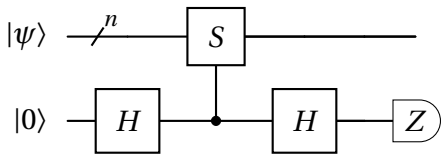


Figure 2: Indirect measurement of the stabiliser S.

We see that the state $|\psi\rangle \otimes |+\rangle$ before the controlled-S gate becomes $|\psi\rangle \otimes |0\rangle + S|\psi\rangle \otimes |1\rangle$ after the controlled operation, and then

$$|\psi\rangle \otimes |+\rangle + S|\psi\rangle \otimes |-\rangle = (\mathbb{1}_{\otimes n} + S)|\psi\rangle \otimes |0\rangle + (\mathbb{1}_{\otimes n} - S)|\psi\rangle \otimes |1\rangle \tag{14}$$

after the H-gate. Measuring the ancilla qubit in the Z-basis then projects the n-qubit system onto the state $(\mathbb{1}_{\otimes n} \pm S)|\psi\rangle$ (up to normalisation), depending on the measurement outcome; this state lives in the respective ± 1 eigenspace of S (cf. Appendix B.1).

Projecting a quantum state onto a stabiliser eigenspace via stabiliser measurements may also be used for logical state initialisation, as will be discussed in Section 2.1.11. Performing X- and Z- stabiliser measurements is of great use to detect Pauli errors in a circuit; Appendix B.2 provides some more details on the concrete implementation of X-type and Z-type stabiliser measurements.

Example 4: Error correction via stabiliser measurement

As an illustration of the error-correcting capabilities implied by Expression 14, we discuss the bit-flip repetition code, which can correct single-qubit X-errors by encoding one logical qubit into three physical ones.
The stabiliser generators of this code are $S_1 = Z_1Z_2I_3$ and $S_2 = I_1Z_2Z_3$. Now assume that the logical plus state $|+\rangle_L = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ suffers from an X_1-error, yielding the system state $|\psi\rangle = \frac{1}{\sqrt{2}}(|100\rangle + |011\rangle)$.
Performing an indirect measurement routine for S_1 (cf. Figure 2) then results in the state

$$(\mathbb{1}_{\otimes 3} + ZZI)\,|\psi\rangle \otimes |0\rangle + (\mathbb{1}_{\otimes 3} - ZZI)\,|\psi\rangle \otimes |1\rangle \tag{15}$$

before measurement. For the given $|\psi\rangle$, the measurement outcome is -1 deterministically, where the system (after normalisation) remains in state $|\psi\rangle$. Now also performing an indirect measurement of S_2, we deterministically measure eigenvalue $+1$ on the respective measurement ancilla, with the system state remaining uncorrupted.

Thus, we know that any potential X-error anti-commutes with S_1, yet commutes with S_2, which lets us conclude that indeed, an X-error occurred on the first qubit. Applying the correction X_1 on the 3-qubit system state $|\psi\rangle = \frac{1}{\sqrt{2}}(|100\rangle + |011\rangle) = X_1\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ then restores the logical plus state $|+\rangle_L$.

Generally, if an error anti-commutes with a stabiliser, then we can catch it via the stabiliser measurement, also known as *error detection*. Assume that a detectable Pauli error $E \in \mathcal{P}_n \setminus \mathcal{C}(\mathcal{S})$ occurred (e.g., the X_1 error in Example 4 above). Then by design, E anti-commutes with at least one stabiliser $S \in \mathcal{S}$. For convenience, we assume that this element is the first generator S_1, i.e., $ES_1 = -S_1E$. Measuring this generator will yield -1 as outcome, which means we have detected the error.

Now in a stabiliser code, the codespace is defined by multiple stabiliser generators (each of which is a n-qubit Pauli string). Repeating the above procedure for each stabiliser generator (potentially on a fresh ancilla) in a *round of stabiliser measurements* projects the system onto a shared eigenspace of *all* stabilisers. Even though this is oftentimes not C (since some measurement outcomes are -1), we thereby effectively discretise any error to an error subspace that differs from the codespace only by Pauli operations; to map the system back to the codespace, we then need to find a set of quantum gates (Pauli operators) that anti-commutes precisely with those stabilisers whose measurement result was -1, while commuting with the ones whose measurement outcome read $+1$. This step is required to transcend mere error detection towards *error correction*, which is crucial to keep a quantum computation alive even when the quantum state is compromised throughout.

Herein, a *decoder* gathers the ± 1-ancilla measurement outcomes (the so-called

measurement syndrome), which is obtained from *syndrome extraction* (the act of indirectly measuring a set of stabiliser group generators), and infers a correction to the state[7]. Note that this error correction method is not restricted to Pauli errors; since the n-qubit Pauli strings form a basis of the n-qubit operator space, any arbitrary error (such as a coherent error caused by an over-rotation of a rotation gate) can be decomposed into a superposition of Pauli strings. Measuring stabilisers then also discretises arbitrary errors into Pauli errors, allowing the decoder to map back to the correct state, given that the error is correctable.

For a stabiliser code, the set of *correctable errors* generally is the union of the elements of $\mathcal{S}$ (which trivially fix any logical state) and some subset of the detectable errors. This is made precise by the following condition.

Theorem 1: Condition for error correction

For a stabiliser code, suppose $\{E_i\}$ is a set of Pauli operators satisfying

$$E_j E_k \notin \mathcal{C}(\mathcal{S}) \setminus \mathcal{S} \;\forall\, j,k. \tag{16}$$

Then $\{E_i\}$ is a correctable set of errors for the code. (cf. [3, Theorem 10.8])

This theorem asserts that any pair of correctable errors must not combine to a logical operator in $\mathcal{C}(\mathcal{S}) \setminus \mathcal{S}$. Since stabiliser operators fix the codespace and are thus certainly correctable, we see that for non-stabiliser elements, being detectable is a necessary (yet not sufficient) condition for being correctable[8].

As a consequence of Theorem 1, logical Pauli errors are not correctable; they map states within the codespace and thus commute with all stabilisers. This motivates us to introduce the *code distance* d, quantifying how resilient a code is against errors.

Definition 1: Code Distance

The code distance is the minimum weight w of all operators in $\mathcal{C}(\mathcal{S}) \setminus \mathcal{S}$, where the weight of a Pauli string is defined as the number of non-identity Pauli operators in its tensor product. (cf. [3, Ch. 10.5.5])

To be precise, in the definition of the code distance above, we only consider the elements of $\mathcal{C}(\mathcal{S})$ that have positive phases, from which we subtract all elements $\pm S$ for $S \in \mathcal{S}$ [20]. This ensures that we don't minimise the weight over elements of the centraliser that are trivial up to global phase.

Intuitively, the code distance tells us that an error has to act non-trivially on at least d qubits in order to fully mimic a logical operation. Given distance d of a $[[n,k,d]]$-quantum code, we can detect arbitrary errors on up to $d-1$, and correct arbitrary errors on up to $\lfloor \frac{d-1}{2} \rfloor$ qubits [3], analogous to classical coding theory [21]. It is clear that we

[7] In particular, the decoder infers a Pauli error e from the syndrome, and determines a correction e' that yields the identity modulo the stabiliser group.

[8] Since the identity (i.e., the case of no error) should certainly be correctable in any reasonable QEC scheme, we require $\mathbb{1} \in \{E_i\}$, from which it follows that for any correctable error $E_1 \in \{E_i\}$, we have $E_1 = \mathbb{1}E_1 \notin \mathcal{C}(\mathcal{S}) \setminus \mathcal{S}$. Therefore, E_1 being correctable implies it to be either a stabiliser element or a detectable error in $\mathcal{P}_n \setminus \mathcal{C}(\mathcal{S})$, proving (the contrapositive of) the necessary condition. Yet, multiplying a detectable error E_2 with some logical operator L, the error $E_3 = E_2 L$ is still detectable, while not both E_2 and E_3 can be correctable (since $E_2 E_3 = L \in \mathcal{C}(\mathcal{S}) \setminus \mathcal{S}$), demonstrating that being detectable is not sufficient for being correctable.

are generally interested in QEC codes with relatively large code distance. For a code of distance d, if the number of physical errors does not exceed the bound of $\lfloor\frac{d-1}{2}\rfloor$ errors, a decoder such as a *Lookup Table Decoder* (cf. Section 5.2.3) is able to correct the discretised (correctable) error reliably to the original quantum state in the logical codespace[9]. The Steane $[[7,1,3]]$-code, for example, can correct any single error on its seven physical qubits. However, if the error correction threshold is surpassed, then the act of trying to correct it will complete a logical error[10]. Note that this is in line with the observation that not all detectable errors are correctable; in particular, beyond the threshold of weight-$\lfloor\frac{d-1}{2}\rfloor$ errors, a correction E_k might map an error E_j back to the codespace, but thereby complete a logical operation $E_jE_k \in \mathcal{C}(\mathcal{S})\setminus\mathcal{S}$ of weight $\frac{d}{2}+\frac{d}{2} = d$, yielding an incorrect state *within* the codespace afterwards[11]. The predicament of completing logical errors will be further elaborated on in Section 5, when discussing the simulation of logical encoding circuits.

Other challenges to error correction include that the decoding of non-Clifford gates (cf. Section 2.1.5) sometimes has to be performed in real-time [6], posing a significant constraint[12]. And also the stabiliser measurements themselves, used for syndrome extraction, can introduce errors into the system [27, 28], which motivates the design of fault-tolerant circuits, as discussed in Section 2.1.10.

From a numerical viewpoint, if we can correct $\lfloor\frac{d-1}{2}\rfloor$ errors and errors do not propagate uncontrollably (cf. Section 2.1.10), then for a logical error to occur, there need to be at least $\lfloor\frac{d-1}{2}\rfloor+1$ errors in the system. Hence, a distance-d code can in principle suppress the physical error rate p to a logical error rate of

$$p_L \propto p^{\lfloor\frac{d-1}{2}\rfloor+1}; \tag{17}$$

it is clear that for sufficiently small noise, we have $p_L \ll p$, which is a central observation in the theory of fault tolerance (cf. Section 2.1.10).

2.1.5 Clifford Formalism

Having introduced the construction of stabiliser codes, their logical Pauli operators and error-correcting capabilities, we next elaborate on the last missing component of Figure 1, the Clifford group $\mathcal{C}_n$. It appears frequently in QEC, since it allows us to capture dynamics in the stabiliser formalism.

Assume that we want to apply some general unitary transformation U on a quantum state $|\psi\rangle$, which is stabilised by a Pauli operator $P \in \mathcal{P}_n$. Reminiscent of the Heisenberg picture in quantum physics [29], we can write

$$U|\psi\rangle = UP|\psi\rangle = UPU^\dagger U|\psi\rangle = UPU^\dagger(U|\psi\rangle). \tag{18}$$

[9] Since decoding generally requires keeping track of many possible error syndromes, many decoding schemes such as *Minimum Weight Perfect Matching* [22, 23], *Maximum Likelihood Decoding* [24] or *Belief Propagation* [25] are designed to infer which (discretised) error occurred with large probability, given a certain syndrome.

[10] We may refer to these large-weight errors as *uncorrectable errors*; in [26], such errors are termed 'harmful errors'.

[11] Note that this is a direct consequence of *error degeneracy* — the fact that many different errors, such as any set of errors differing only by logical operators, can lead to the measurement of the same syndrome.

[12] Here, different decoders are often tailored towards certain classes of codes, and differ in their scalability and accuracy.

Reading Equation 18 from right to left, it becomes clear that the new state $U|\psi\rangle$ is a $+1$-eigenstate of the operator $UPU^\dagger$. In other words, an update of the state can be expressed via an update of its stabilisers, captured by conjugation action. This change of perspective has an immense benefit for computational efficiency, as will be discussed shortly.

Example 5: State update via conjugation action

Consider $|\psi\rangle = |0\rangle$ and $P = Z$ for $n = 1$. Applying a Hadamard gate to our state gives $H|0\rangle = |+\rangle$, which is stabilised by X. At the same time, the conjugation action of a Hadamard is uniquely determined by the expressions

$$HXH = Z \text{ and } HZH = X, \tag{19}$$

so capturing the update in the state can also be captured by updating the stabiliser $Z \mapsto HZH = X$.

It is then natural to ask which unitaries U satisfy $UPU^\dagger \in \mathcal{P}_n$ and thus preserve the Pauli group under conjugation. Unsurprisingly, these unitaries are precisely the Clifford operators – elements of the Clifford group $\mathcal{C}_n$ on n qubits.

Definition 2: Clifford group

The Clifford group $\mathcal{C}_n$ is defined as the normaliser of $\mathcal{P}_n$ within the entire space of unitary transformations U^{2^n}, i.e.,

$$\mathcal{C}_n = \mathcal{N}_{U^{2^n}}(\mathcal{P}_n) = \{U \in U^{2^n} : UPU^\dagger \in \mathcal{P}_n \quad \forall P \in \mathcal{P}_n\}. \tag{20}$$

For fixed n, the Clifford group $\mathcal{C}_n$ is a finite group, generated most prominently by the set

$$\left\{H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \mathrm{S} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, CX_{i\to j} = |0\rangle\langle 0|_i \otimes \mathbb{1}_j + |1\rangle\langle 1|_i \otimes X_j\right\}. \tag{21}$$

In the light of error-correcting codes, let's consider a set of $n-k$ generators of $\mathcal{S}$ as well as some k logical operators $\{L_j\}_{j=n-k+1}^{n}$ that stabilise a logical state $|\psi\rangle_L$. All these stabilisers are Pauli operators. Importantly, the canonical commutation relations allow us to express all elements in $\mathcal{P}_n$ (up to global phase) via $2n$ binary variables, encoding the physical X_i and Z_i operators on each qubit i[13]. To capture a stabiliser state, then, we can draw upon a so-called *tableau*, in which one records the n stabilisers of $|\psi\rangle_L$ via binary entries. Mathematically, we herein exploit the isomorphism between the quotient group $\mathcal{P}_n/\langle i\mathbb{1}_{\otimes n}\rangle$ (under multiplication) and the $2n$-dimensional vector space $\mathbb{F}_2^{2n}$ (under addition). Note that in general, we write n binary vectors of size $2n+1$, where in addition to the physical X_i and Z_i operators, we store the phase ± 1 of each operator in binary form [30]. We can contrast the space complexity of storing these $\mathcal{O}(n^2)$ entries with that of storing $\mathcal{O}(2^n)$ coefficients of a general quantum state.

[13] This suffices as representation, since it gives rise to the operators $\mathbb{1}_i$ and Y_i: if both X_i and Z_i are 0, we encode $\mathbb{1}_i$, and if both are 1, we encode Y_i [30].

Example 6: Steane code tableau

The six Steane code stabilisers from Example 2 can be collected in the below tableau.

	X							Z							ϕ
	X_1	X_2	X_3	X_4	X_5	X_6	X_7	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	
S_1	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
S_2	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0
S_3	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
S_4	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0
S_5	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0
S_6	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0

(22)

This determines the codespace of the Steane code; every state $|\psi\rangle \in C$ is a joint $+1$-eigenstate of all the Pauli strings encoded by the rows of the above tableau. If we add a row for the logical $Z_L = Z_2Z_4Z_6$ operator to Tableau 22, it captures the logical zero state $|0\rangle_L$ of the Steane code.

A tableau allows us to keep track of the updates to the stabilisers in a logical computation. But what update needs to be performed to the tableau in Example 6 when we apply a logical Hadamard operator to our stabiliser state $|\psi\rangle_L$, for example? By the conjugation action of the physical H we saw earlier, applying a Hadamard on the i^{th} qubit amounts to swapping the i^{th} and $(n+i)^{\text{th}}$ column of the tableau (corresponding to swapping the entries for X_i and Z_i, respectively). Knowing that the logical Hadamard for the Steane code can be written as $H_L = H^{\otimes 7}$, from the Z_L operator in Example 6 we can then determine the logical X operator as $X_L = X_2X_4X_6$. Comparing this to the canonical logical $X_L = X_3X_5X_6$ operator described in Example 3, we find that they indeed only differ by the stabilisers S_2 and S_3, and thus belong to the same coset in the quotient group of logical operators. In reference to [30], Figure 2 in [31] provides a concise visualisation of how Clifford gates act on a given tableau.

It turns out that each of the Clifford operations can be captured by a so-called symplectic matrix [8], providing us with the tools of linear algebra to update stabiliser tableaus efficiently, building upon the conjugation picture discussed above. For example, the column swap update caused by a Hadamard gate can be expressed as a permutation matrix! The resulting computational efficiency of working with Clifford operators is famously captured in the Gottesman-Knill theorem.

Theorem 2: Gottesman-Knill theorem

Any quantum computer performing only: a) Clifford group gates, b) measurements of Pauli group operators, and c) Clifford group operations conditioned on classical bits, which may be the results of earlier measurements, can be perfectly simulated in polynomial time on a probabilistic classical computer. (cf. [29])

This theoretical result asserts that as long as we only perform Clifford operations, we don't need a quantum computer, but can simply keep track of the dynamics on a classical computer. In particular, the computational complexity of Clifford (or stabiliser) circuits is polynomial in the number of qubits [30] rather than exponential as for general unitary circuits. And even any kind of measurement can be emulated in

polynomial time, where trading off some spatial complexity with computational complexity (by not only storing stabilisers, but also so-called *destabilisers*) further reduces computational cost [30]. This becomes surprising when observing that Clifford operations are capable of producing highly entangled quantum states. As an example, Clifford gates alone allow for the creation of GHZ states, or performing protocols like quantum teleportation. So despite its restrictive definition, implying that only a vanishingly small fraction of all possible quantum states can be reached via Clifford operations[14], we find that $\mathcal{C}_n$ constitutes an immensely useful set of operations. Consequently, a plethora of research has investigated how to generate and classically simulate Clifford circuits efficiently [32, 33, 34], also targeted to specific hardware constraints [31]. This has lead to the development of powerful stabiliser circuit libraries such as `QuantumClifford` [35] or `Stim` [36].

Furthermore, given some tableau for a target state, there exist methods to synthesise a stabiliser circuit that maps to the respective state [30, 37, 38], about which we will have more to say in Section 2.1.11. In the context of quantum architectures with restricted connectivity, [31] investigates a Steiner tree approach to choose pivots for a Gaussian elimination process on the final tableau (where the Clifford circuit becomes the inverse of the elimination operation), [39] employs Graph Networks, and [40] repurposes *Google DeepMind's FunSearch* algorithm to create a Large Language Model-driven evolutionary search for quantum state preparation circuits. Also, [26] proposes a Reinforcement learning approach for circuit discovery. In the scope of this study, we work exclusively with Clifford circuits, which are versatile enough for our task of encoding logical $|0\rangle_L^{\otimes k}$ states.

2.1.6 Canonical Form

It is time to broaden our toolbox regarding the tableau formalism and logical operators $X_{L,i}$ and $Z_{L,i}$. This section treats the Gottesman canonical tableau, which lays the foundation for the construction of unitary encoding circuits in Section 2.1.11.

We saw that a stabiliser code is defined via the stabiliser subgroup $\mathcal{S}$, which in turn can be expressed in tableau form, consisting of the physical X- and Z-terms in the left and right half of a matrix, respectively (cf. Example 6). Since $\mathcal{S}$ can be generated by many different choices of stabilisers, Gottesman introduced the so-called *standard form* of a tableau [17, Ch. 4.1], which is a unique canonical tableau form for a stabiliser code.

To obtain this form, we start with some tableau containing a set of stabiliser generators of the quantum code at hand, expressed in matrix form. We then perform Gaussian elimination on the X-submatrix and permute the columns to obtain a matrix of the form

$$\begin{array}{c} \\ r \\ s \end{array} \begin{pmatrix} \overbrace{}^{r} & \overbrace{}^{n-r} & \overbrace{}^{r} & \overbrace{}^{n-r} \\ \mathbb{1}_r & A & B & C \\ 0 & 0 & D & E \end{pmatrix}, \tag{23}$$

where A, B, C, D and E denote submatrices of the indicated dimensions[15]. Next,

[14] In fact, the number of pure stabiliser states on n qubits scales as $2^{[1/2+o(1)]n^2}$ [30], compared to the uncountably infinite number of non-stabiliser states.

[15] We omit phases for ease of notation here.

perform Gaussian elimination on the submatrix in rows $r+1:r+s$, permute the columns further and eliminate submatrices to obtain the standard form depicted below.

Concept 1: Gottesman standard form

The Gottesman canonical form of a tableau is given by

$$\begin{array}{c} \\ r \\ s \end{array} \left(\begin{array}{ccc|ccc} \overbrace{}^{r} & \overbrace{}^{n-k-r} & \overbrace{}^{k} & \overbrace{}^{r} & \overbrace{}^{n-k-r} & \overbrace{}^{k} \\ \mathbb{1}_r & A_1 & A_2 & B & \mathbf{0}_{r\times s} & C \\ 0 & 0 & 0 & D & \mathbb{1}_s & E \end{array} \right), \tag{24}$$

where A_1, A_2, B, C, D and E again denote the submatrices of indicated dimensions.

A few remarks are in order: Firstly, note that in Matrices 23 and 24, r denotes the rank of the X-part of the tableau, and s denotes the rank of the submatrix in rows $r+1:r+s$. Here, we assume that the initial tableau contains $n-k$ non-degenerate stabilisers (i.e., that the stabilisers form a minimal generating set of the code), such that we have $n-k=r+s$. This assumption comes at no loss of generality, since we may either remove any redundant generators a priori, or ignore all-zero rows (corresponding to redundant stabilisers) in Matrix 24 after performing Gaussian elimination.

Secondly, note that when performing Gaussian elimination on the X-submatrix, we need to apply the row operations on the entire matrix. In particular, swapping rows i and j swaps the entire row of the matrix[16].

Lastly, observe the potential need to also perform columns swaps, i.e., qubit permutations, after each of the two Gaussian eliminations. Since the tableau matrix captures the X- and Z-part of each stabiliser separately, any column permutation in one submatrix needs to be applied to the other submatrix as well. For example, when swapping the first and second column of the X-submatrix of a tableau, the respective columns also need to be swapped in the Z-submatrix.

From the requirement that all logical operators commute pairwise, and that any logical operator commutes with all stabilisers[17], we may further express the logical X-operators as

$$[X_L] = \left[\mathbf{0}_{k\times r} \quad E^T \quad \mathbb{1}_k | C^T \quad \mathbf{0}_{k\times n-r}\right]. \tag{25}$$

Abiding by the same commutation constraints, and additionally satisfying the canonical commutation relations for logical Pauli operators, the canonical logical Z-operators are of the form

$$[Z_L] = \left[\mathbf{0}_{k\times n} | A_2^T \quad \mathbf{0}_{k\times s} \quad \mathbb{1}_k\right]. \tag{26}$$

The derivation of the above results can be found in Appendix B.5. For an example of how this canonical form relates to the examples of the Steane code encountered in previous examples, and how it can be constructed in `julia`, please refer to Appendix B.6. Note that Expression 26 tells us that we can always choose a set of logical Z_L-operators consisting exclusively of physical Z-operators. This foreshadows a convenient feature of CSS codes, which are discussed in the next section.

[16] Phase information equally has to be accounted for in those row operations.

[17] Note that we omitted phase information altogether in this treatment, which however does not alter the commutation relations.

2.1.7 CSS Codes

In the following, we introduce a popular family of stabiliser codes, the Calderbank-Shor-Steane (*CSS*) codes. This class of codes has many favourable properties and includes most of the well-known quantum codes, such as the *Surface Code* [41] or *Bivariate Bicycle Codes* [19]. CSS codes derive from classical coding theory, using the parity check matrices dicussed in Section 2.1.1: Let C_1 and C_2 be two classical binary linear codes, encoding k_1 and k_2 bits, respectively, in n bits each. Further, let H_1 and H_2 be the (binary) parity check matrices of C_1 and C_2, such that

$$C_1 = \left\{c \in \mathbb{F}_2^n : H_1 c = 0\right\} \text{ and } C_2 = \left\{c \in \mathbb{F}_2^n : H_2 c = 0\right\}, \tag{27}$$

and require that the dual code

$$C_2^\perp = \left\{c' \in \mathbb{F}_2^n : \langle c'|c\rangle = 0 \forall c \in C_2\right\} \tag{28}$$

is a subset of C_1, i.e., $C_2^\perp \subseteq C_1$. Next, suggestively relabel the indices 1 and 2 to Z and X: Write $H_X = H_2$ for the parity check matrix of $C_X = C_2$, and likewise $H_Z = H_1$ and $C_Z = C_1$, such that $C_X^\perp \subseteq C_Z$. Then the matrix

$$\begin{bmatrix} H_X & 0 \\ 0 & H_Z \end{bmatrix} \tag{29}$$

is the tableau of a $[[n, k_x + k_z - n]]$ quantum CSS code, containing its stabiliser generators as rows[18]. As discussed in Appendix B.8, the above tableau verifies

$$H_X H_Z^T = \mathbf{0}, \tag{30}$$

as required for commutativity of stabilisers.

The form of Tableau 29 tells us that for CSS codes, the X-type and Z-type stabilisers can be separated cleanly for a particular choice of generators. Note that for the Steane code (which is a CSS code), we can witness this feature in Tableau 22. In particular, the Steane code can be constructed using the Hamming-$[7,4]$-code from Example 1 as the classical seed for both the X- and Z-submatrix, which yields Tableau 22 up to Z-type stabiliser multiplication. Owing to their special stabiliser structure, we may simplify the Gottesman canonical form of CSS codes to read

$$\mathrm{T} = \begin{matrix} r \\ s \end{matrix} \left(\begin{array}{ccc|ccc} \overbrace{}^{r} & \overbrace{}^{s} & \overbrace{}^{k} & \overbrace{}^{r} & \overbrace{}^{s} & \overbrace{}^{k} \\ \mathbb{1}_r & A_1 & A_2 & & \mathbf{0}_{r\times n} & \\ & \mathbf{0}_{s\times n} & & D & \mathbb{1}_s & E \end{array}\right), \tag{31}$$

where the commutation relation on H_X and H_Z in Equation 30 determines

$$\mathbb{1}_r D^T + A_1 \mathbb{1}_s + A_2 E^T = \mathbf{0}_{r\times s} \Leftrightarrow D = E A_2^T + A_1^T. \tag{32}$$

Furthermore, the logical X-operators reduce to

$$[X_L] = \begin{bmatrix} \mathbf{0}_{k\times r} & E^T & \mathbb{1}_k | \mathbf{0}_{k\times n} \end{bmatrix},$$

[18] Further details on the codespace dimension CSS codes can be found in Appendix B.7.

which implies that we can choose logical operators without mixing X- and Z-types. This will be of importance for the fault-tolerant implementation of logical encoding circuits in this work, and it also allows us to express the code distance of a CSS code as

$$d = \min\left\{d_X, d_Z\right\}, \tag{33}$$

where d_X and d_Z are the respective distances of the X- and Z-type operators. In particular, based on Definition 1, these distances can be determined as the minimum weight of the (non-trivial) logical X- and Z-operators, respectively, which [19] captures with the expressions

$$d_X = \min\left\{|X_L| : X_L \in \ker(H_Z) \setminus \mathrm{rs}(H_X)\right\} \text{ and } d_Z = \min\left\{|Z_L| : Z_L \in \ker(H_X) \setminus \mathrm{rs}(H_Z)\right\}. \tag{34}$$

For d_X, a logical X-operator lives in the kernel of H_Z, such that it commutes with all Z-type stabilisers, reduced by the (X-type) elements of the stabiliser group $\mathcal{S}$, which live in the row space of H_X, $\mathrm{rs}(H_X)$. Note that in the above formulation, we have $\ker(H_Z) = C_Z$ (since H_Z is the parity check matrix of C_Z), and $\mathrm{rs}(H_X) = C_X^\perp$ (since H_X is the parity check matrix of C_X, and thus the generator matrix of $C_X^\perp$, cf. Appendix B.8), which gives rise to an alternative notation sometimes used in literature [9, Ch. 4.4][20]. The construction of d_Z is entirely analogous.

2.1.8 qLDPC Codes

Recently, quantum Low-Density Parity Check (*qLDPC*) codes have gained increased attention [42]. While qLDPC codes need not be CSS codes [43], many interesting examples at the intersection of the two code families exist, such as Hypergraph Product codes [44] and Bivariate Bicycle (BB) codes [19, 45]. These codes are rooted in classical LDPC codes [46], and are defined via the weight of their stabilisers as well as the number of interactions of each qubit with the stabilisers.

Definition 3: qLDPC codes

"A [qLDPC] code is a family of stabilizer codes such that the number of qubits participating in each check operator and the number of stabilizer checks that each qubit participates in are both bounded by a constant." (cf. [42, II.A.2])

This characteristic is favourable for repeated stabiliser measurements in syndrome extraction, for example, and has established qLDPC codes as promising candidates for scalable fault-tolerant quantum computing. Importantly, such codes can be used for high-rate encodings, where multiple logical qubits are defined on a set of physical qubits in a code block [47]; in particular, qLDPC codes commonly encode more than one logical qubit ($k > 1$) per logical patch [48, 49, 50], whereas the planar surface code, for example, usually encodes a single logical qubit [42].

Generally, qLDPC codes are based on non-local stabilisers, necessitating long-range interactions on a quantum architecture [44], as well as the design of tailored high-rate logical operation [47, 51]. These aspects motivate the study of adapted distributed implementations of these codes, as discussed in Section 2.2.

Out of the many variants of qLDPC codes, in this work we focus our attention on *Multivariate Bicycle* codes [52], and in particular, Trivariate Bicycle [53, 52] and Bivariate Bicycle [19, 48, 54, 55] codes. These codes belong to the overall category of Multi-

variate Multicycle qLDPC codes [50], and are CSS codes[19]. Mutivariate Bicycle codes are constructed by building up Matrix 31 in a systematic way. In the following, we discuss this construction for the Trivariate Bicycle code, from which the Bivariate Bicycle case will become clear as well.

Firstly, define the cyclic shift matrix S_i as the $i \times i$ matrix that has a single 1 in row k and column $k+1 \mod i$ for $1 \leq k \leq i$. For example, we have

$$S_3 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}. \tag{35}$$

Next, fix parameters l and m, and define the three variables ('Trivariate')

$$x = S_l \otimes \mathbb{1}_m, \quad y = \mathbb{1}_l \otimes S_m \quad \text{and} \quad z = S_l \otimes S_m, \tag{36}$$

each of which describes a matrix of size $lm \times lm$. Observe that by the design of S_i, we have $x^l = y^m = \mathbb{1}_{lm}$ [19], i.e., we find two cycles ('Bicycle'): x is l-cyclic and y is m-cyclic. The periodicity of z equals the least common multiple of l and m, since z is not an independent variate[20]. The three variates x, y and z commute pairwise [52]. Now construct matrices A and B as polynomials in x, y and z, where addition is performed element-wise and modulo 2. For example, set $A = \mathbb{1}_{lm} + x^3 + y$ and $B = x + y^3 + z^2$. Exponents exceeding the periodicity of the respective variate reduce according to the cyclic property. Then the number of terms in each of the sums, which is referred to as the *weight* of the matrix in [52], is upper bounding the number of nonzero entries in each of the rows and columns of the matrices A and B, respectively (with equality holding if the polynomials are chosen such that there are no cancellations of terms).

With this setup in place, we can finally construct our stabiliser matrix in Equation 31 with

$$H_X = [A|B], \quad H_Z = [B^T|A^T]. \tag{37}$$

Note that this construction directly verifies the necessary commutation relation:

$$H_x H_Z^T = [A|B][B^T|A^T]^T = [A|B] \begin{bmatrix} B \\ A \end{bmatrix} = AB + BA = 2AB = \mathbf{0}_{lm \times lm} \tag{38}$$

where we have used that A and B commute and addition is binary. Further note that the number of nonzero entries in each row and column is upper bounded by $w = \text{weight}(A) + \text{weight}(B)$, such that the weight w plays the role of the upper bound constant on the number of qubits supporting each stabiliser, and the number of stabilisers any single qubit interacts with, as outlined in Definition 3.

From this construction, it follows that a Trivariate Bicycle code thus is a code on $n = 2ml$ physical qubits. Since A and B are symmetric, we find that the stabiliser matrix also has $2ml$ rows. However, there usually is redundancy in the rowspace, i.e., the stabilisers defined by the above construction do not generally form a minimal generating set of $\mathcal{S}$. Thus, the dimension of the codespace, which is the dimension of the kernel of the stabiliser matrix, is dependent on the specific polynomials used, and can be comparably large; given a large ratio k/n, we speak of a *high-rate* quantum code.

[19] Note that the structure of CSS codes tells us that the weight of each row and column of the submatrices H_X and H_Z in Matrix 31 is bounded for qLDPC-CSS codes.

[20] We could make z independent by choosing to work with a Trivariate Tricycle code.

Example 7: Trivariate Bicycle-[[12,2,3]] code

Setting $l = 2, m = 3$ and

$$A(x,y) = x + y^2, \quad B(x,y) = \mathbb{1}_{12} + z^4, \tag{39}$$

we obtain a Trivariate Bicycle codes on $2 \cdot 2 \cdot 3 = 12$ qubits. This code defines 2 logical qubits (two of the twelve stabilisers are redundant) and has distance 3. The stabilisers of this [[12,2,3]] code may be found in Appendix B.11 for reference.

For completeness, note that Bivariate Bicycle codes are constructed entirely analogous to the Trivariate Bicycle codes discussed above, with the only difference that we ignore the variable z in Equation 36 when constructing matrices A and B.

2.1.9 Universal Quantum Computing

We introduced $\mathcal{P}_n$ and $\mathcal{C}_n$ as important groups in the theory of QEC (cf. Figure 1), and also already saw an example of a logical Clifford operator (the logical Hadamard in Section 2.1.5). Such logical Clifford operators can always be constructed from physical Clifford operators [56]. However, we would be remiss not to also mention what lies outside of the Clifford group; as we have noted, $\mathcal{C}_n$ can be simulated classically, so any true power of a quantum computer needs to stem from non-Clifford operations. Thus, a central concept in quantum computing, which carries over from the physical qubit to the logical space, is the notion of *universality*.

Definition 4: Definition: Universality

A "set of gates is said to be *universal for quantum computation* if any unitary operation may be approximated to arbitrary accuracy by a quantum circuit involving only those gates" [3, Ch. 4.5].

For example, the set including *all* single qubit gates and the CX gate is universal, but there also exist discrete sets that can arbitrarily approximate any unitary, such as

$$\{H, T, CX\}.$$

As alluded to above, however, a universal gate set cannot consist of Clifford gates alone. In general, we need the Clifford group and at least one non-Clifford gate (such as the T-gate in the discrete set above) in our toolbox to obtain universality, defining a trade-off between expressability and efficiency of quantum circuits.

To achieve universality also of logical operations for a given stabiliser code, we rely on elaborate methods that synthesise and inject non-Clifford logical gates into our system. This is often done via so-called *magic state distillation*, which has been proven to be sufficient for universal quantum computing when combined with Clifford operations [57]. The idea behind this is to outsource all non-Clifford operations to *T-gate factories* [48] or *magic state distillation factories* [58], in which non-Clifford states are distilled and then used to apply the action of the non-Clifford gate (T-gates or arbitrary Pauli rotations in general [59]) onto the code module. On the code-module itself, then, only Clifford operations need to be performed. Another possible approach to

achieve universality is to utilise *code switching*, which combines two quantum codes "with complementary sets of transversal gates by transferring encoded information between them" [60].

While it is clear that we cannot track non-Clifford states with the tableau mechanism described earlier, it is important to note that logical Clifford gates are often much simpler to implement *fault-tolerantly* for a given code, which will be discussed in the next section. Therefore, separating code modules and magic state factories [48] has the advantage of simplifying the instruction set complexity on the code modules. In general, the encoded state after a magic state injection will still be an encoded state in C, such that we don't lose the error-correcting capabilities of the code. Related to our work, logical zero states are a building block of magic state distillation (cf. Section 3); besides from this motivation, however, we will not further investigate logical non-Clifford operators or universality.

2.1.10 Fault Tolerance

In order for a quantum computation to be robust against noise, we wish to perform operations that are fault-tolerant — a notion frequently appearing in QEC [61, 62, 63, 64, 26, 60]. Even though the precise definitions of fault tolerance vary, intuitively, it refers to the property of some procedure (such as code state preparation, a quantum gate or measurement) to not allow errors to spread in an uncontrolled way. For example, in Figure 3, a single X-error on the control qubit spreads to two errors, which may become problematic if we cannot counteract such spread accordingly.

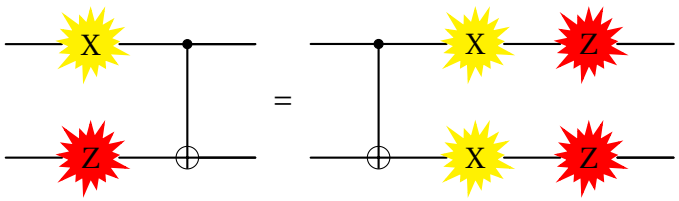


Figure 3: X-and Z-Error propagation through a CX gate: X-errors (Z-errors) propagate via the control (target) of the gate, where raw and propagated errors are marked in the same color, respectively. Visualisation performed using the `Quantikz` package [65].

The definition that we employ in our study of fault tolerance makes the notion of restricted error propagation precise. In the following, we write $t = \lfloor \frac{d-1}{2} \rfloor$ for a distance-d QEC code.

Definition 5: Fault-Tolerant Procedure

"For a code correcting up to t errors per block, a procedure is fault-tolerant if any pattern consisting of at most $s \leq t$ faults during the procedure, together with at most $r \leq t - s$ pre-existing errors on each input block, produces at most a correctable number of errors on each output block, that is, at most t errors per output block, and does not alter the encoded logical action." (cf. [9, Definition 13.1])

While Definition 5 captures that errors don't accumulate, in the study of fault-tolerant encoding circuits, we will later focus our attention on the faults "during the procedure",

since the notion of "pre-existing errors" is not relevant in this context. For encoding circuits, we thus adapt the definition put forth in [27, Definition II.3]. Furthermore, we restrict our attention to CSS codes, for which the X- and Z-error sets can be treated separately [27]. Similar to [66, Definition 4.2], we thus require Definition 6 to hold separately for X and Z errors (cf. [67]).

Definition 6: Fault-Tolerant State Preparation for CSS codes

A logical state preparation circuit is fault-tolerant to distance $d = 2t+1$[a] if for every $s \leq t$, any s errors within the circuit propagate to no more than s X-errors, and to no more than s Z-errors, at the end of the circuit. (cf. [27, 66])

[a] The code distance is the minimum of d_X and d_Z, as discussed in Section 2.1.7.

Note that in the above definitions, error counts within the procedure refer to (combinations of) error locations, while the error count at the end of the circuit refers to the minimal weight of all equivalent errors (including logical operators not stabilising the target state, cf. Section 5) under multiplication with the stabiliser group. Here again, the weight of an operator is defined as its Pauli support (cf. Definition 1). For the remainder of this work, we refer to the latter type of count as *error weight*.

Apart from the count-based definition of our error model, the notion of fault tolerance can also be viewed through the lense of error probabilities. In Section 5, we will discuss a circuit-level noise setup that consists of qubit initialisation, single- and two-qubit gate, idle and measurement noise with varying error probabilities. For a theoretical argument, however, we may assume independent errors that occur with a uniform probability p to simplify matters [6]. Given such noise, if some procedure abides by Definition 5, then we expect the logical error rate of the procedure to follow

$$p_L = Cp^{t+1}, \tag{40}$$

where we made the proportionality in Equation 17 explicit with some proportionality constant C. Depending on the code and architecture, the constant C captures, for example, the number of possible combinations of error location that produce an uncorrectable weight-$(t+1)$ error [6]. Importantly, as the physical noise rate p decreases (e.g., by improvements in gate fidelities), there will be a noise level p_{th} at which Equation 40 reduces to $Cp_{\text{th}}^{t+1} = p_L = p_{\text{th}}$. Below this *noise threshold*, the logical error rate is actually smaller than the physical error rate. In view of such noise thresholds, *Google Quantum AI* recently demonstrated a sub-threshold memory experiment on their Surface-97 code [68], which entailed preserving a logical qubit for multiple QEC cycles.

While sub-threshold error rates are useful in and of themselves, the inequality $p_L < p$ can be used to derive the *Threshold Theorem*, as proved for example in [69], [70], and [71]. This theorem asserts that at the cost of a gate overhead that scales polylogarithmically in the number of error locations in the circuit, we can achieve arbitrarily small error rate [6]. This remarkable result tells us that, in principle, QEC can be leveraged to enable large scale quantum computation with high precision. The central idea behind the constructive theorem is that for sub-threshold error probabilities, one can define a QEC code on encoded logical qubits recursively, in a process termed *code concatenation*. A precise version of the Threshold Theorem can be found in Appendix B.3.

Enabling a fault-tolerant quantum computation entails making each part of the

computation, from logical encoding over logical gates and error correction up to measurement, fault-tolerant. At the level of logical operations, we find that Clifford operations can oftentimes be performed fault-tolerantly. In particular, many codes allow for the application of transversal (bitwise) physical Clifford operations [72, 73] to perform the corresponding logical operation, making the procedure fault-tolerant by design [3, Ch. 10.6]. For the Steane code, the entire Clifford group can be implemented transversally [3, p. 10.6], and CSS codes in general exhibit "transversal CNOT, Hadamard, Pauli gates and logical measurement[s] in Z or X basis" [45]. To make the notion of transversality somewhat more tangible, consider a logical CX-gate between two Steane codes (cf. Appendix B.4), which relies on physical CX-gates between the corresponding qubits in each code block, where any physical error causes at most one error *in each encoding block*. Performing operations between different encoding blocks, or code modules [48], in such a way foreshadows the discussion of distributed QEC codes in Section 2.2. While transversality is sufficient but not necessary for fault tolerance, it is usually a favorable property, simplifying the description and implementation of logical operations. However, Eastin and Knill showed that in general, it is not possible to have a transversal universal gate set [74]. Therefore, universal fault-tolerant computing remains an active area of research [75, 59, 76, 60].

While software-hardware co-design is needed to leverage the potential of QEC methods for large-scale quantum computation, the principle of fault tolerance will be central to this quest. In this work, we focus on the very first step in the fault tolerance chain — the encoding of a logical qubit, as discussed in the next section.

2.1.11 Logical State Encoding

Having explored the construction of stabiliser codes as well as central notions of quantum error correction, we return back to a detail we have omitted so far — the actual preparation of logical states. It is conceptually straightforward to prepare the physical all-zero state $|0\rangle^{\otimes n}$ on a quantum computer: Simply measure all qubits in the Z-basis and perform the necessary Pauli-X corrections (or wait for a time much longer than the relaxation time T_1 of the qubits). But in order to perform algorithms on logical qubits, i.e., encoded blocks of physical qubits, we need to somehow map our n-qubit system into the 2^k-dimensional codespace C. Broadly speaking, there exist two approaches to achieve this goal: measuring the stabilisers of the logical state, or performing an (unitary) encoding circuit that produces the state. In the following, we give an overview over both procedures, also with respect to considerations of fault tolerance.

Measurement-Based Encoding. One way to map a quantum system to the code space of a given quantum code is to measure all stabilisers in a generating set of $\mathcal{S}$. Furthermore, if we want to generate a specific (Clifford) state of the code, we can measure the $n-k$ generators plus the k logical operators that uniquely determine the state. As discussed in Section 2.1.4, the resulting state then lives in some shared ± 1 eigenspace of $\mathcal{S}$, since each measurement collapses some initial state to the eigenspace of the stabiliser being measured [3, Ch. 5.8]. We then require decoding and Pauli corrections to map to the joint $+1$-eigenspace; however, these corrections are often not needed in real-time, since we can redefine the codespace to whichever syndrome outcome the initial measurement yielded, in a process referred to as *Pauli frame tracking* [6]. As long as all operations are Clifford, decoding can then be understood as a purely classical task,

which can occur after the quantum computation has finished. For CSS codes specifically, based on the state we wish to encode, we also only need to measure one type of stabilisers; for example, to create the logical $|0\rangle_L$ state starting from the physical all-zero state, we only need to measure the X-type stabilisers[21] [27, 28]. In Section 2.1.4, we already saw how a stabiliser measurement can be implemented as an indirect measurement via an ancillary system, depicted in Figure 2. The measurement of logical operators that fully specify the state can be performed in the same way [3, Ch. 10.5].

Overall, this way of encoding is very elegant; by construction, we synthesise the target encoded state, no matter which initial state we hold in our register [8, Ch.13.1]. However, note that any measurement error, or equivalently any idling error that occurs on the qubits during the stabiliser measurements, yields inconsistent syndrome information. This is aggrevated by the fact that the measurements themselves can lead to errors on the qubits[22]. Employing fault-tolerant measurements (to ensure that we correct, or at least detect, the potential errors introduced by the stabiliser measurements) is in general resource-intense, however. In particular, it relies on additional verified ancilla states [8, Ch. 12.1], or d rounds of stabiliser measurements, with d being the code distance, as depicted in [77, Figure 1][23].

A current topic of research is thus *single-shot error correction* [50], where one round of stabiliser measurements during syndrome extraction suffices to maintain fault tolerance. This approach can be applied to logical state preparation as well under suitable circumstances [28].

We see that it is in general possible to realise state preparation via the (indirect) measurement of stabilisers and logical operators. Furthermore, recurring syndrome extractions for decoding operations require stabiliser measurements throughout the entire quantum operation, necessitating some measurement infrastructure. However, fault-tolerant measurement-based state preparation generally incurs large overheads in the number of ancillas or measurements, and is subject to on-going research.

Circuit Encoding. In this work, we opt for the alternative option of encoding quantum states — via an (unitary) *encoding circuit* that encodes some physical quantum state into its logical counterpart [78, 17, 26]. Equipped with the Gottesman canonical form and logical operators, we can build a quantum circuit that encodes a logical basis state, step-by-step. Such encoding circuits always exist, and [78] as well as [17, Ch. 4.2] provide the first algorithms to encode a computational basis state $|c_1 c_2 \dots c_k\rangle$, $c_i \in \{0,1\}$ into a stabiliser codespace. A working algorithm to synthesise the state preparation circuit of a logical basis state, based on the Standard Form 24, can be found in Appendix B.9, where Algorithm 3 is presented alongside its derivation. It is noteworthy that every stabiliser code exhibits encoding circuits for basis states that consist entirely of Clifford gates [62, Ch. 3.5].

Broadly speaking, Algorithm 3 encodes a computational basis state on k qubits into its logical counterpart on n qubit by first creating the embedded basis state $|0\rangle^{\otimes n-k} \otimes |c_1 c_2 \dots c_k\rangle$ on n qubits, then applying logical X-operators and lastly projecting to the

[21] This follows since the Z-type stabilisers stabilise the initial state, and the stabiliser measurements commute by construction.

[22] For an example, consider the Z-error occurring on the ancilla qubit in Figure 16, which is propagated onto the data qubits.

[23] This yields a layered syndrome over time, which could be regarded as a 'spatio-temporal code distance'.

code space. By slightly changing the algorithm, we can also encode an arbitrary quantum state $|\psi\rangle$ on k qubits (i.e., a superposition of computational basis states) at the expense of a slightly increased gate count. For this, assume that we have a quantum register of the form $|0\rangle^{\otimes n-k}\otimes|\psi\rangle$, where $|\psi\rangle$ is some arbitrary quantum state on k qubits (cf. [61, Figure 8]). Dropping the tailored application of CX gates based on the value of c_i in Algorithm 3, and instead applying the corresponding CX gates with controls on all qubits $n-k+i:n$, it follows by linearity that the resulting state after the projection is the encoded $|\psi\rangle_L$. The adaption required for this can be found as Algorithm 4 in Appendix B.9 as well. On the other hand, $|\psi\rangle_L$ can also always be created by first encoding some logical computational basis state, and then applying the corresponding logical operation mapping $|0\rangle_L$ to $|\psi\rangle_L$.

CSS Code Circuit Encoding. For CSS codes, Algorithm 3 simplifies: Firstly, we don't need to apply a phase gate after Hadamard gates, since the Z-part of the tableau has been eliminated for the first r rows. Secondly, all Z-controlled Pauli operations are CX gates, for the same reason. Thus, the required gate set for CSS code encoding circuits is $\{H, CX\}$, assuming that all stabilisers have positive phases[24]. Furthermore, for the synthesis of the logical $|0\rangle_L^{\otimes k}$ state, only the X-type stabilisers need to be considered, similar to what we saw in the stabiliser measurement approach. Algorithm 1 captures this simplified version, and Figure 4 displays the encoding of the Steane code resulting from the CSS code-tailored algorithm.

Algorithm 1 Logical $|0\rangle_L^{\otimes k}$ state preparation circuit from Gottesman canonical form for positive-phase CSS codes. Based on [17, Ch. 4].

Input: (i) Quantum register input state $|0\rangle^{\otimes n}$, (ii) Canonical Tableau T in fully reduced form 31
Output: Encoded logical $|0\rangle_L^{\otimes k}$ state on n qubits

```
for i = 1 : r do
    apply H(i)
    for j = (r+1) : n do
        x = T_{i,j}
        if x = 1 then
            apply CX(i, j)
        end if
    end for
end for
```

The problem of synthesising efficient encoding circuits for CSS codes is framed as a latin rectangle problem in [79]. Building upon this, [80] provides an optimisation method exploiting overlap between stabilisers to encode the logical $|0\rangle_L$ state of the $[[23,1,7]]$-Golay CSS code. In a similar vein, [81] optimises the circuit size of a general encoding circuit for multiple codes, via circuit equivalence rules.

The authors of [27] introduce an automated synthesis framework for CSS code state preparation circuits based on satisfiability solving (SAT). And by the Clifford property of encoding circuits, the Clifford circuit synthesis method in [32] can be utilised to

[24] The Steane, Shor, Bivariate and Trivariate Bicycle codes considered in this study fulfill this; otherwise, we only need to add single-qubit X- and Z-correction gates.

obtain encoding circuits for QEC codes as well. The circuits extracted from these algorithms are not necessarily optimal, and targeted at a general monolithic quantum architecture, not specifically taking into account connectivity constraints. In this context, [31] investigates synthesis under hardware constraints, based on Steiner trees. In a similar vein, [26] employs Reinforcement Learning (RL) for circuit synthesis of CSS and non-CSS codes, operating on a distance metric between the target and current tableau at every step of the learning process. And [82] leverages RL to discover "QEC codes and their encoding circuits for a given gate set, qubit connectivity and error model, from scratch". In our work, we extend this hardware-aware approach by adapting the overlap method from [80] to a DQC setting, supplemented with a Genetic Algorithm, to compile efficient state preparation routines for logical $|0\rangle_L^{\otimes k}$ states; details of this are discussed in Section 4.2.

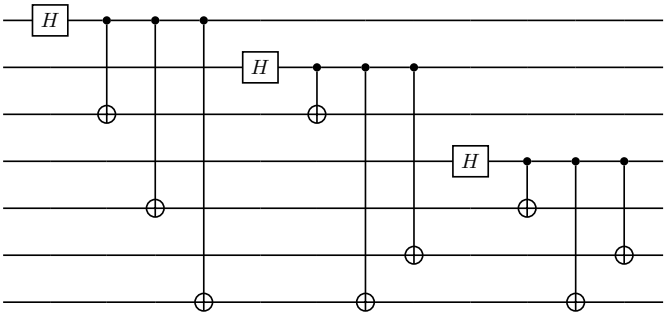


Figure 4: Encoding of the $|0\rangle_L$ state for the Steane $[[7,1,3]]$ code, using Algorithm 1.

Fault-Tolerant Encoding Circuits. An encoding circuit maps a physical state to the codespace, yet this process is not automatically fault-tolerant. Similar to the complications that arise for measurement-based state preparation, quantum circuits can propagate errors to higher-weight errors, thereby violating fault tolerance, as hinted at in Section 2.1.4. In this context, research has been devoted to making encoding circuit fault-tolerant by adding a verification step after the circuit [80, 63, 26, 27, 67]. Such verification analysis has also appeared under the name *flag* fault tolerance [83, 84, 85]. The central idea of this approach is to add a set of stabiliser measurements via which the uncontrolled propagation of errors can be caught, or 'flagged' (taking into account that the verification circuit itself needs to be constructed in a fault-tolerant manner). Such fault-tolerant encoding circuits were dicussed for the Steane code [63], the $[[23,1,7]]$-Golay code [80], the Symplectic Double code [49] and a number of other small codes [26, 27], where a common objective is to keep the verification overhead as small as possible. We will make use of such verification circuits in Section 5.

Depending on the circuit complexity, verification information can be used to discard flagged runs, or to apply the corresponding correction gate. Such correction based on a verification circuit is even possible without actually performing the measurement of the ancilla used in the stabiliser 'measurement': The authors of [86] introduce a measurement-free auxiliary system that actively performs error correction based on the errors propagated through the two-qubit gates of an indirect stabiliser measurement circuit (cf. [86, Figure 6]). And alternatively to the verification approach, [87] hand-crafts a method to fault-tolerantly encode the logical zero state of the concatenated Shor-$[[9,1,3]]$ code without needing measurements or ancillas in the fault-tolerant encoding procedure.

Importantly, these works focus mostly on the fault-tolerant encoding of the $|0\rangle_L$ or $|+\rangle_L$ states of their respective codes. As alluded to above, by linearity of unitary operations, with encoding circuits we are generally not restricted to the computational basis states. Broadly speaking, however, it is much more difficult to make encoding circuits for general quantum states fault-tolerant via verification circuits. In the context of non-Clifford states, the central reason for this is that logical non-Clifford operators cannot be measured easily (for verification purposes), and that measurement of a logical Clifford (which is more straightforward to perform) leads to a collapse of the non-Clifford state. Alternatively, the authors of [63] investigate fault-tolerant encoding of a particular magic state for the Steane code, using fault-tolerant measurements. But even within the Clifford regime, if an encoding circuit has the capability to encode any arbitrary state (per the construction in Algorithm 4, for example), its very structure often easily introduces high-weight or logical errors via propagation. As an example, a single X-error before the CX-operations in Algorithm 4 directly propagates to a logical X-error. To combat this, one would have to not only measure stabilisers, but also logical operators in the verification circuit, which generally also collapses an arbitrary input state.

Logical state encoding has been demonstrated in real-world experiments for small codes. For example, [88] reports the operation of a Steane code on a 10-qubit ion trap processor (including state preparation, logical operations and syndrome extraction). Figure 10 therein shows a fault-tolerant encoding circuit including verification, as discussed earlier, which is repeated up to three times (since post-selection needs to be performed based on the outcome of the verification measurement). Here, the authors also encode a logical magic state, but explicitly state that they "cannot use the FT encoding circuit used for the logical Pauli states, because the verification step requires the measurement of a logical operator which would collapse the T-state" [88] – the predicament we encountered above. In this context, [58] uses non-fault-tolerant encoding circuits for arbitrary states, akin to Algortihm 4, to encode and then distill magic states on a neutral atom architecture. Also, [89] uses the fault-tolerant encoding circuit for the Steane code based on [63], and a verification-based encoding circuit for the quantum Reed-Muller $[[15,1,3]]$ code; then the authors employ the two verified states as the initial logical states for a code switching protocol on a trapped-ion device, which enables the creation of a magic state with very high fidelity. In [67], the logical zero state of the Golay $[[23,1,7]]$ code is synthesised via flag gadgets on an ion trap device; there, the achieved logical *State Preparation and Measurement* (SPAM) error rate is lower than the physical SPAM error, within a 95% confidence interval.

Throughout this section, we encountered logical state encoding as important building block and starting point for QEC. While both stabiliser measurements and encoding circuits allow for such encoding, resource efficiency (possibly under restricted hardware connectivity) and fault tolerance are important metrics by which we seek to evaluate such approaches. In Section 4, we will optimise CSS code state preparation circuits of the logical $|0\rangle_L$ state for a DQC setting. And in Section 5, we then use a verification circuit approach for fault-tolerant encoding based on [27]; this toolbox specifically draws upon the property that for CSS codes, we can treat X- and Z-type stabilisers separately, examples of which will be provided in Section 5. We next introduce DQC as a promising paradigm for scalable quantum computing.

2.2 Distributed Quantum Computing

While Quantum Error Correction is concerned with protecting quantum information to make error-prone computations more reliable, Distributed Quantum Computing has emerged in order to provide a pathway to scalable quantum computing. As such, DQC offers an alternative to monolithic (from 'mono' = 'one' and 'lithos' = 'stone', referring to a single quantum chip) computing by defining an interconnected network of Quantum Processing Units (QPUs). This can become relevant in order to parallelise quantum algorithms [90], enable quantum-centric supercomputing [91], or scale up systems despite hardware constraints, such as wiring problems [92]. As of summer 2026, many of the leading actors in quantum computing and policy have identified DQC as a viable path towards scalable quantum computing [93, 94, 95].

Recent years have seen much research in this field, investigating how to connect multiple QPUs, also referred to as *modules*[25] or *cores*, with networking links, and how to optimally partition and route qubits over this network. The development of DQC in the context of quantum computing, and research in quantum communication over quantum networks, are thus mutually influential. Networking simulators such as *QuantumSavory* [96] and *NetSquid* [97] allow us to investigate such networking dynamics for DQC simulations.

In the following, we briefly introduce central aspects of performing quantum algorithms on distributed architectures[26]. In particular, we discuss how to distribute a quantum circuit over a network in Section 2.2.1, and how DQC compilation aims to optimise this process in Section 2.2.2. In Section 2.2.3, we introduce ion traps with photonic interconnects as a hardware candidate for potential near-term modular architectures. For a more detailed treatment, the authors of [98] provide an overview over the different layers — from physical over network up to the application layer — that are required to enable DQC.

2.2.1 Quantum Gate Teleportation and Circuit Cutting

As alluded to before, DQC is enabled by harnessing quantum networks. In particular, as the authors of [45] state, any "truly distributed architecture will involve some form of quantum communication between nodes". In these networks, which are naturally designed for communication tasks, we find both classical and quantum connection between nodes. Here, the quantum connection serves to establish entanglement in the form of shared EPR pairs, such as the first Bell state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, between so-called *communication qubits* of different nodes, whereas the classical links enable the exchange of measurement results for conditional local, i.e., intra-core, operations [99].

These quantum network connections then ultimately allow us to perform remote, i.e., inter-core, operations on the *data qubits* hosted on each QPU, such as *state teleportation* or *gate teleportation* [98]. In this work, we focus on circuit distribution via quantum gate teleportation, or *telegates* [100] — a protocol to perform a CX (or other two-qubit) gate between two different cores. The telegate protocol is depicted in Figure 5 and discussed in Appendix B.10; it was originally proposed in [101] (with slightly different gate ordering), and is thus sometimes also referred to as *EJPP protocol*, owing

[25] Note that also the terms *distributed* and *modular quantum architecture* are often used interchangeably.

[26] Similarly to Section 2.1, this shall serve as an introduction to DQC as it relates to this project, and should not be seen as a comprehensive review.

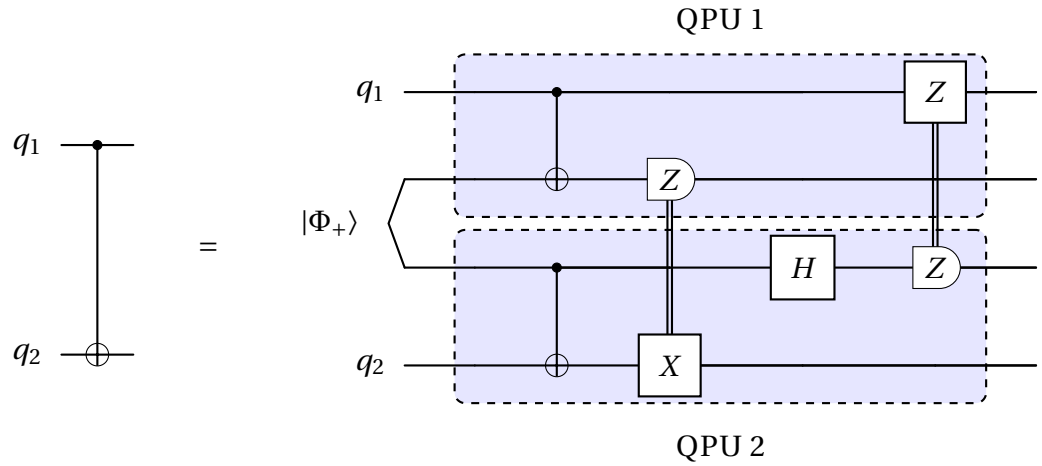


Figure 5: Telegate protocol to perform a remote CX gate between qubits q_1 and q_2 on QPUs 1 and 2, consuming a maximally entangled Bell state on the communication qubits $q_{1,\text{comm}}$ and $q_{2,\text{comm}}$, as derived in Appendix B.10. Adapted from [103].

to the surnames of the authors. As Figure 5 shows, performing a telegate necessitates the establishment of a shared Bell pair between two communication qubits of the two modules involved in the operation. Importantly, this remote-entanglement resource, also referred to as *e-bit* [102], is consumed within the protocol via measurement of the communication qubits. The classical (non-local) communication serves to apply conditional Z- and X-gates, based on the measurement outcomes of these qubits. And the CX gates between data and communication qubits are local operations.

The creation of high-fidelity Bell pair entanglement is one of the core challenges of DQC [98], since it is often orders of magnitude less reliable than other quantum operations [100, 45]. For example, [100] reports a Bell state entanglement fidelity of around 97% between two trapped-ion modules, which is much noisier than local operations on many state-of-the-art machines [5, 104]. Hence, the minimisation of such telegates is the central objective of many DQC compilation techniques, as will be seen in Section 2.2.2. There also exist techniques to increase the fidelity of the noisy interconnects at the cost of producing multiple shared Bell states in a process termed *entanglement purification* [105]; the trade-off between fidelity and execution time arising from performing this process is however not further investigated in this work.

Alternatively, there exist methods that enable a distributed execution of quantum algorithms without having to rely on the quantum connection between the nodes; instead, classical communication for the coordination of distributed execution needs to be in place then. For example, one can split a circuit to effectively distribute it, which is referred to as *wire cutting* when splitting in a temporal manner, and *gate cutting* when splitting in a spatial manner [98]. In either case, the (results of the) subcircuits are recombined after execution, which is known as *circuit knitting*. The authors of [106] investigate optimal partitioning of these cuts for circuit knitting, and `Qiskit` exposes functionality facilitating the implementation of circuit cutting [107]. The drawback of these techniques is that circuit cutting induces an exponential sampling overhead growing with the number of cuts [98].

In both schemes, we find that extending a quantum architecture from a monolithic device to multiple cores bears the promise of parallelism and scalability, while

also introducing challenges regarding exponential sampling overhead (circuit cutting) or costly inter-core quantum communication (quantum networks). Such trade-offs are characteristic for distributed systems. Indeed, the notion of DQC has a classical counterpart, where multiple GPUs are connected to perform a resource-intense computation. This has contributed to the feasibility of Large Language Model training, for instance [108]. Also in the classical setting, the parallelism gained from using multiple cores comes at a cost — the need for meticulous coordination of tasks, defining another trade-off between parallelism and communication overhead. In Section 5, we will investigate to what extent we can perform operations in the DQC regime, and whether high-fidelity encoding of logical states is possible on distributed architectures.

2.2.2 DQC Compilation

Given some shared entanglement resource between quantum cores, the DQC execution of a quantum circuit requires a *compilation* strategy. From an algorithmic viewpoint, the task of finding a compiled version of a noisy circuit that achieves high fidelity with a target state (e.g., the logical state of some QEC code) can be broken down into two sub-tasks. Firstly, the virtual qubits of the circuit need to be mapped to a given hardware architecture. This allocation, or *qubit mapping*, should be performed considering the real qubit connectivity, and take into account the reduction of costly operations (such as telegates) in a first compilation pass. Secondly, the operations of the quantum circuit need to be scheduled and executed respecting further hardware constraints such as a limited native gate set, the availability of communication qubits at each step, or heterogeneous quantum networks. Importantly, in distributed quantum architectures both the intra- and inter-core connectivity can vary, superimposed by chip design and networking capabilities. This execution, or *qubit routing*, problem has analogues in classical traffic flow optimisation, for example.

The mapping and routing components are to some degree interdependent, since mapping provides the foundation for routing operations, while different mappings might be favourable depending on the routing strategy. Thus, the two concepts are frequently subsumed under the term *quantum circuit compilation*.

The DQC research community has introduced various approaches to improve quantum compilation. An early work by Andrés-Martínez et al. express the interaction graph of some circuit as a hypergraph, and consequently investigates the allocation problem as a combinatorial *hypergraph partitioning* problem [102]. In particular, the authors optimise the number of e-bits used for execution by minimising the number of cuts of the hypergraph over its partitions. In a follow-up work, a grouping of gates is included in order to increase the number of remote gates that can be performed with one e-bit [109]. The authors of [110] introduce an algorithm utilising a forward-looking interaction graph, whereas [111] provides a different mathematical flavour by recasting compilation as a *Quadratic Unconstrained Binary Optimisation* (QUBO) problem. Other techniques include using *Steiner trees* [109] or *Hungarian Qubit Assignment* [112] for optimisation of distributed circuits. The authors of [113, 114] and [103] combine the optimisation of quantum gate and quantum state teleportation in a graph-based approach. Besides these mathematically-inspired analytical approaches, various authors propose (Deep) Reinforcement Learning [115, 116, 117] or Evolutionary Algorithms [118] to optimise the execution of circuits on DQC architectures. Also *Quantum Architecture Search* (QAS) has been employed for DQC applica-

tions, e.g., in [119] and [120]. Throughout, several authors consider holistic hardware-aware approaches including explicit networking conditions [121, 122, 109, 123], rather than treating the minimisation of telegates as such.

Overall, a small number of reviews compares different compilers and compilation strategies, notably [112] and the aforementioned [98]. Still, compilation of quantum circuits on distributed quantum architectures is a field of active research, and no single unified framework exists so far (especially when incorporating realistic quantum network constraints in the consideration). Importantly, a succesful compilation (which reduces the number of e-bits required to execute a circuit, for example) does not automatically yield a circuit that is fault-tolerant. The task of compiling fault-tolerant quantum circuits generally necessitates the investigation of error propagation through telegates, and the addition of gadgets that counter-act harmful spread accordingly, which adds another layer of complexity. In our work, we optimise the performance of fault-tolerant encoding circuits in a DQC setting via tailored circuit optimisation. Here, we uphold fault tolerance as design principle, motivating our choice of optimisation objective. This ultimately allows us to integrate distributed verification circuits, mitigating harmful error propagation, even in the DQC setting[27].

2.2.3 DQC Hardware

Considering that no single quantum hardware has established itself as the leading platform for quantum computing at large so far, it is unsurprising that also multiple candidate platforms for modular architectures exist. A review of such platforms for DQC can be found in [124, p. II. C.]. One example of a promising DQC hardware platform are ion traps with photonic interconnects [7, 100, 125]. These systems usually consist of multiple QPUs, each hosting a number of data qubits[28] and communication qubits, as well as a photonic quantum network; Figure 6 provides a sketch of a potential DQC architecture. Importantly, while these systems transcend monolithic ion trap system by adding a quantum network for remote entanglement, ion traps themselves are currently among the leading quantum computing platforms in terms of gate fidelities and connectivity [5].

When performing a telegate with the protocol depicted in Figure 5, the remote entanglement between communication qubits is created by entangling each of them with a photon, routing the photons to a beam splitter and letting them interfere. The process of entangling ions and photons is outlined in [126], for example. As discussed in [100], a particular click pattern at the detectors can then herald successful entanglement. The authors of [100] utilise different ion species for data and communication qubits (Ca^+ and Sr^+, respectively), where the data ion exhibits favourable coherence times, and the communication ion is apt for interfacing with the photonic network. Furthermore, they also employ auxiliary qubits to enhance the mixed-species local operations resulting from this design choice. With this setup, [100] undertakes an experimental realisations of such a photonically connected ion trap system, demonstrating the ability to entangle communication qubits across QPUs with 97% fidelity, and consequently perform a telegate with 86% fidelity[29].

[27] Details of this will be discussed in Sections 4 and 5.

[28] The authors of [100] refer to data qubits as *circuit qubits*, but it should be noted that in the general setting we describe, data qubits can also act as ancilla qubits for distributed stabiliser measurements in the context of QEC, for example.

[29] Note that this telegate is a controlled-Z gate, whereas Figure 5 shows a controlled-X gate.

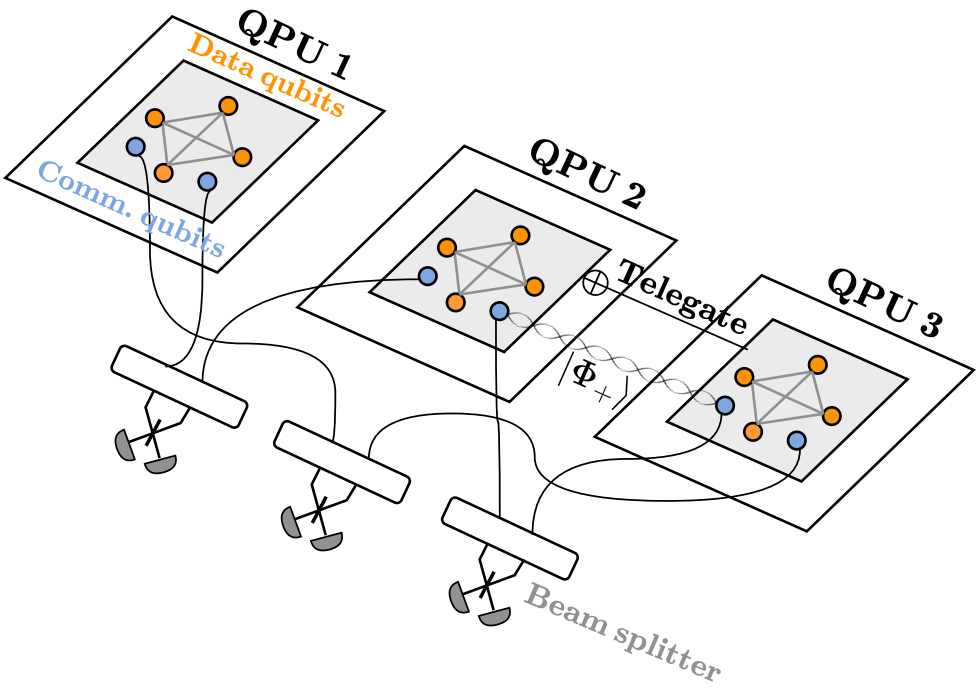


Figure 6: Sketch of ion traps with photonic interconnects as potential DQC architecture. Communication qubits are assumed to be fully connected to all data qubits on their respective QPU (connections between qubit species and classical links are omitted to avoid clutter). Adapted from [100, Figure 1].

For the simulations carried out in this work, we draw upon ion traps with photonic interconnects for motivating our simulation setup and noise channels, as discussed in Section 5. However, we will not further examine the physical layer of a DQC architecture, including the implementation of a gate set via lasers for trapped ions, or high-fidelity photon transport, for example.

2.3 QEC meets DQC: Distributed Stabiliser Codes

The present work lies at the intersection of QEC and DQC, for which we introduced central notions in the previous sections. Here, we provide a brief overview of existing work realising QEC methods specifically on distributed architectures. An instructive classification for this can be found in [127], which differentiates approaches based on the level of modularity. In particular, the authors discuss three types of architectures (Type-I to Type-III), which vary in the number of qubits each module hosts, but also in the qualitative way the modules are utilised in the context of QEC. This classification will also guide our review in the following paragraphs.

In Type-I architectures, we typically find a large number of modules, each hosting a small number of data and communication qubits [127]. While this generally ensures high intra-core connectivity, it also requires frequent e-bit generation between nodes for syndrome extraction as well as for logical operations. Recent works investigate toric surface codes distributed over a system of modules consisting of one to two data qubits, a few auxiliary qubits for stabiliser measurements, and one communication qubit each [77, 128]. In particular, both works investigate the role of distributed

GHZ state creation to enable remote gates between the cores: In [77], distributed stabiliser measurements rely on the creation of 3- or 4-qubit GHZ states (for the case of two or one data qubit, respectively), and [128] performs noise simulations specific to color centers in diamonds to investigate memory decoherence accumulating on data qubits during GHZ state creation idling time. In this context, research on non-local stabiliser measurements for Type-I systems dates back more than a decade, with [105] providing a protocol for quantum computing on "very noisy [quantum] networks".

For Type-II architectures, one distributes "large quantum error-correcting codes across multiple quantum modules" [127], where inter-core operations are enabled by shared entanglement resources, as discussed in Section 2.2. Here, the size of a module is larger than in Type-I systems. An exemplary work is [125], which distributes a Floquet over multiple cores, as depicted in [125, Figure 1]. Since this code exhibits weight-2 stabilisers only, the number of telegates required for syndrome extractions is limited. This is beneficial for the memory experiment performed, where one aims to maintain a logical state (without applying logical operations) in the presence of noise. The authors of [129] investigate a distributed color code in a Type-II fashion, and in particular, error thresholds for different decoders in this distributed setting. In both works, the underlying noise model distinguishes between intra- and inter-core noise, with inter-core error rates being elevated. In [130], the Bivariate-Bicycle $[[144,12,12]]$ code is distributed over 12 modules, motivated by ion trap architectures, with flying qubits mediating interactions.

Lastly, Type-III systems host a full error-correcting code on each module, whereas the quantum network connecting the modules is enabling the performance of logical operations. This allows, for example, the application of "fault tolerant computations between nodes [...], such as transversal gates, distributed lattice surgery procedures, or teleportation of logical states" [127]. Therefore, Type-III architectures are promising candidates for fault-tolerant QEC on DQC. The authors of [45] assess non-local operations and fault-tolerant teleportation of the surface code and qLDPC codes on such architectures under circuit-level noise. And also in [48], logical operations are performed using entire code blocks, where a Bivariate Bicycle qLDPC code with $k = 12$ logical qubits per module provides for a "long-range-connected modular hardware, [...] in contrast to the conventional surface code architecture based on short-range, monolithic hardware".

Overall, the distinction between the three types is not always sharp; for example, [127] lists [131] as a Type-II architecture — there, the authors connect multiple full surface code patches via 'seams' that implement telegates and therefore experience larger error rates, without directly supporting transversal operations. Therefore, a distinction should not only concern the number of qubits that each QPU hosts, but also the types of fault-tolerant operations that each type of architecture allows for.

Concept 2: Type-II systems

In this work, we focus primarily on the capability of Type-II systems to enable the distribution of a larger code across modules of modest qubit size, with a limited number of communication qubits per module.

The sample architecture in Figure 6 aligns with the above Type-II narrative, assuming that all of the architecture's data qubits together define some QEC code.

3 Project Outline

DQCode builds on top of a plethora of research, extending existing literature in various directions. Multiple authors investigate encoding circuits for QEC codes on monolithic architectures [17, 132, 81], including connectivity constraints [87, 26] and fault tolerance considerations [80, 87, 26, 27, 49, 133]. Other works examine QEC on distributed architectures in general [77, 45, 131], and also on Type-II architectures [125, 129]. Yet, to the best of our knowledge, no research treats encoding circuits on Type-II architectures specifically. And while DQC compilation methods [109, 103, 134] can generally be used as an optimisation primitive for such a task, they require a baseline encoding circuit to be compiled, and have not been applied to the problem at hand (especially not in a fault tolerance context). We summarise this in the problem statement below.

Problem Statement 1

Existing research investigates (i) unitary encoding circuits for QEC codes on monolithic architectures, and separately (ii) other QEC routines, such as memory experiments, on Type-II distributed architectures. Fault-tolerant encoding circuits for qLDPC and other codes on Type-II architectures remain unexplored, even though they can constitute a building block for QEC on near-term distributed systems.

This creates research opportunity in the optimisation and evaluation of fault-tolerant encoding circuits for Type-II systems, for both of which we may heavily draw upon existing literature. As such, the following project statement is a direct response to the gap in the current research landscape.

Project Statement 1

DQCode optimises and evaluates fault-tolerant logical $|0\rangle_L^{\otimes k}$ encoding circuits for Multivariate Bicycle qLDPC codes on distributed Type-II architectures.

As such, DQCode combines quantum circuit search as well as benchmarking of noisy circuits at the intersection of QEC and DQC. In the remainder of this section, we provide motivation for these particular choices of research directions, and formulate concrete research questions.

3.1 Motivation

Establishing the general research direction, our project statement implicitly comprises multiple research choices. Therefore, we discuss our motivation to (i.) investigate encoding circuits as preparation method for logical $|0\rangle_L^{\otimes k}$ states, and (ii.) choose Multivariate Bicycle qLDPC codes as code instances in the Type-II DQC architecture framework.

(i.) Encoding Circuits for Logical $|0\rangle_L^{\otimes k}$ State Synthesis. In Section 2.1.11, we paid particular attention to logical state encodings, for which we saw two alternative approaches — measurement-based initialisation and circuit encoding. Both approaches

require overhead to make them fault-tolerant, be it via fault-tolerant stabiliser measurements or verification circuits, respectively. We expect the additional overhead of repeated measurements for fault tolerance to be quite large in a distributed setting, whereas the tool provided by [27] provides a way to find efficient verification routines for unitary encoding circuits. Despite choosing encoding circuits as the primary object of interest in this study, we conduct a resource comparison of our method with the alternative of repeated distributed stabiliser measurements in Section 5.3.

We saw in Algorithm 4 that encoding circuits can in general encode arbitrary states. However, as discussed in Section 2.1.11, making a general encoding circuit fault-tolerant for an arbitrary input state cannot easily be achieved by the means of a verification circuit as introduced in Section 2.1.10. In order to prioritise fault tolerance in our method, we choose to encode the logical all-zero state, thereby making our work compatible with the verification circuit gadgets implemented by [27] (cf. Section 5.1). This further allows us to tailor the optimisation of the circuit structure to a DQC setting, as discussed in Section 4.2. Our particular target logical state can also serve as a primitive for other routines, such as magic state distillation [3, Ch. 10.6] or code switching [60], and verified $|0\rangle_L$ states may be used as a building block of the *Steane EC scheme* for syndrome extraction [9, Section 13.2].

Furthermore, mapping to the logical all-zero state (and thus to the codespace) fault-tolerantly is the starting point of large-scale fault-tolerant quantum computing. Here, using a fault-tolerant encoding circuit enables one to perform (fault-tolerant) logical operations right after the encoding, without the need for syndrome extraction or decoding[30]. This follows from Definition 6: For a fault-tolerant encoding circuit of a distance d-code state, any set of $s \leq t = \lfloor (d-1)/2 \rfloor$ errors propagates to an error of weight $\leq s$. Then for a posterior logical operation satisfying Definition 5 of fault-tolerant procedures, such weight-s errors remaining after the encoding circuit remain correctable also after the logical operation, even if $r \leq t - s$ new errors occur during the operation.

(ii.) Multivariate Bicycle Codes on Type-II DQC Architectures. In Section 3, we saw that the high-rate qLDPC codes constitute a promising family of codes due to their constant-weight stabilisers. Herein, the choice of Multivariate Bicycle codes at the intersection of qLDPC and CSS codes is motivated by their additional property of separable X- and Z-type stabilisers, which is a convenient feature for the construction of verification circuits [27].

Furthermore, Type-II architectures are viable candidates for near-term distributed quantum computing. We already mentioned the challenge of scaling monolithic quantum chips; as a remedy, Type-II systems allow us to define a large code on multiple devices of comparably small size. To illustrate the usefulness of such architectures for near-term experiments, note that a monolithic architecture requires n data qubits on a QPU in order to implement a $[[n,k,d]]$ code; for example, the $[[144,12,12]]$ qubit code discussed in [48] requires a QPU with at least 144 qubits. Disregarding the additional overhead of ancilla qubits, the latest superconducting devices (such as the `IBM Heron` processor with 156 qubits[31]) could in principle host the data qubits of this code, while many ion trap systems (such as the `Quantinuum Helios-1` computer with 98 qubits [5]) cannot yet offer the same capacity. In contrast, a Type-II architecture can

[30] While this provides motivation for our study, the construction of such logical operations is not within the scope of our work.

[31] `https://quantum.cloud.ibm.com/docs/en/guides/processor-types` (accessed 08.08.2026)

accomodate for any QPU size, at the overhead of adding communication qubits; in particular, we could distribute the data qubits of the 144-qubit code evenly on twelve cores of moderate qubit size (twelve data qubits plus a small number of communication qubits). The ability to connect multiple ion traps with photonic interconnects (as an example of a Type-II DQC system) thus drastically reduces the size constraint on each QPU.

In comparison to other DQC models (cf. Section 2.3), the communication overhead in Type-II here is generally smaller than in Type-I systems, since more qubits are connected within each core. Furthermore, since a code is distributed over multiple cores, the size of a single QPU is not limiting the choice of the QEC code being used (as exemplified above with a high-rate qLDPC code), whereas for Type-III architectures, each core typically hosts a full code. Generally speaking, using small modules produces large overhead in communication but can lead to near-term experimental realisations, while larger modules are more difficult to implement, yet often come with fault-tolerant logical operations, e.g., via transversality. Type-II architectures cover some middle ground; they allow for the realisation of larger codes on near-term machines, where the size of a single QPU does not restrict the code size, while reducing the extensive communication overhead of Type-I architectures[32]. As discussed in a recent work [135], the distribution of a code in a Type-II fashion can additionally benefit the toleration of device failures in multi-core systems.

In terms of code-architecture co-design, many authors investigate local codes (such as Floquet code [125], color codes [129] or surface codes [131]) on Type-II systems, where only a few non-local stabilisers span across boundaries of cores. In contrast, the high-rate Mutlivariate Bicycle qLDPC codes we consider generally exhibit low-weight stabilisers that require long-range interactions between qubits. Our motivation herein is to investigate whether a careful mapping of qubits, combined with an optimisation of the circuit structure for minimised remote interactions, can effectively accommodate for the non-locality associated with qLDPC code computation. Within our scope of logical state preparation, the overarching goal therefore is to establish Type-II systems as viable candidates for near-term implementations of qLDPC codes. This would allow advances in research on photonic interconnects, for example, to directly contribute to the viability of qLDPC implementations, as alternative to efforts for enabling monolithic long-range interactions[33].

3.2 Research Questions

The `DQCode` pipeline we create based on the above motivation comprises both the optimisation of encoding circuits for multiple Type-II architectures in Section 4, and the DQC simulation of these optimised circuits with added verification circuits in Section 5. Our central focus lies on Multivariate Bicycle codes, as discussed above. In these subsequent sections, we will answer the following research questions underlying `DQCode`.

1. Given a QEC cycle-aware mapping (cf. Section 4.1.3), can the number of required

[32] Yet, efficient fault-tolerant logical operations tailored to codes on Type-II architectures remain an open research question [125].

[33] In [136], for example, researchers introduce a layered chip design combined with a new family of *barbell codes* to enable qLDPC codes on superconducting hardware, which is typically constrained by nearest-neighbour interactions.

telegates on a Type-II architecture be significantly reduced over the canonical encoding from Algorithm 1 by applying DQC circuit compilation and other optimisation methods in the context of quantum circuit search? If so, which of the considered methods are most effective to this end?

2. How can a verification circuit be used in a Type-II distributed setting to ensure fault tolerance of state encoding in view of Definition 6? To what extent can we provide numerical evidence of fault tolerance via DQC simulations?

3. Can we find a 'measurement-free' encoding (requiring no stabiliser measurements) similar to [87] in the DQC setting?

4. How do the intra-core error rate p and the Bell state initialisation error rate p_{Bell} impact the logical state initialisation error rate (LIER) (cf. Section 5.2.4) of the $|0\rangle_L^{\otimes k}$ state in circuit-level noise simulations? Do we find noise floors for very small error rates?

5. Are there pseudothresholds, at which the LIER falls below p? If so, how do p and p_{Bell} relate at those relative to one another, and to error rates attainable by state-of-the-art or near-term machines?

6. What impact does the number of cores in a modular Type-II architecture have on the optimisation and simulation results?

7. For which codes and architectures are encoding circuits or measurement-based initialisation favourable in terms of resource utilisation, and how does requiring fault tolerance change this comparison?

Throughout the remainder of this work, we shall see how answering these questions also poses new questions for future research opportunities.

4 Optimised Synthesis of State Preparation Circuits for Type-II Architectures

> "The first step to a fault-tolerant gadget for stabilizer states is generally to do a non-fault-tolerant encoding circuit to create the state."
> – Daniel Gottesman, [8, Ch. 13.1]

We have already encountered methods that synthesise encoding circuits for stabiliser codes, and in particular CSS codes, such as Algorithm 1. In this section, our main focus lies on the optimisation of encoding circuits for the Trivariate Bicycle-$[[12,2,3]]$ and Bivariate Bicycle-$[[18,4,4]]$ qLDPC codes, as discussed in the previous section. For comparison, we additionally include the Steane-$[[7,1,3]]$ and Shor-$[[9,1,3]]$ code, which are more widely explored in literature, and the Color-$[[17,1,5]]$ code, which serves as an example of a code with distance $d > 3$[34]. Here, the term *optimisation* transcends mere compilation, and rather refers to quantum circuit search[35] with the particular objective of reducing inter-core communication for a fixed mapping. Note that such optimisation does not produce fault-tolerant circuits a priori. Nonetheless, by the very choice of optimisation objective, it serves as a stepping stone towards fault tolerance, reducing the telegate count of encoding circuits in a way that allows us to apply verification circuit techniques, which will be outlined in Section 5.

In the following, we discuss preliminary considerations in Section 4.1 and various optimisation methods and baselines in Section 4.2. We then present the results thereof in Section 4.3, followed by limitations and possible future avenues of work in Section 4.4.

4.1 Optimisation Preliminaries

Before we dive into the optimisation methods employed for circuit synthesis, we frame the optimisation problem in Sections 4.1.1 and 4.1.3, and lay out our assumptions on the system setup in Sections 4.1.2 and 4.1.4.

4.1.1 CSS Code Encoding Circuit Synthesis

In Algorithm 1, we saw that for CSS codes (with positive phases) we can always construct an encoding circuit that consists entirely of H and CX gates. Furthermore, the number of Hadamard gates in the circuit is fixed by the number of X-type stabilisers of the canonical tableau of the target state [27]. For example, the Steane code requires three Hadamard gates (cf. Figure 4). Here, the Hadamard gates can be pushed to the first layer of the circuit, which allows us to interpret them as $|+\rangle$ state initialisations. From this perspective, the task of finding an efficient encoding circuit reduces to the task of finding an efficient CX circuit after physical state initialisation in the Z- and X-basis. As a direct consequence for our optimisation methods, when we define a gate set to optimise over, it suffices to choose CX gates[36]. In a DQC setting, this

[34] Please refer to Appendix B.11 for a summary of the tableaus used in this study.

[35] In literature on parameterised quantum circuits (PQCs), quantum circuit search is referred to as *Quantum Architecture Search*[137]; our search concerns non-parameterised circuits.

[36] For our Monte Carlo Tree Search, we will include Hadamards in the gate set as well, as outlined in Section 4.2.3.

optimisation task can be rephrased to minimising the gate count of an encoding circuit by minimising both the number of telegates between cores, and the number of intra-core CX gates. Due to the disproportionately higher overhead associated with telegates (requiring remote entanglement and fast classical processing of mid-circuit measurements, for example), these two optimisation objectives will appear with different weight in the objective functions discussed in Section 4.2. Note that we restrict our attention to quantum gate teleportation, and leave the incorporation of quantum state teleportation (i.e., the teleportation of a state to another QPU) [111, 103], for future work. This allows us to cleanly separate between data and communication qubits in our DQC simulations, contrary to [103], where additional routing overhead is needed to free up communication qubits before they can be entangled to mediate remote gates again (cf. [103, Figure 7]). Focusing on gate teleportation might even be desirable in order to exploit strengths of different ion species for data and communication qubits [100] without overhead. Furthermore, all optimisation in this section refers to gate optimisation, aiming at reducing telegate count; as shown in [27], it is also possible to optimise for depth as an alternative objective. However, we deem the effect of reducing the number of remote qubit interactions more relevant in our Type-II context.

When searching for an efficient encoding circuit, there exists much freedom derived from the choice of stabilisers already. A naïve estimate of the size of the search space for a CX circuit of depth d (given a fixed qubit mapping, as discussed in Section 4.1.3) is

$$\left[\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \frac{n!}{(n-2i)!i!}\right]^d, \tag{41}$$

where we count the number of possible combinations of control and target qubits for each layer. For example, for $n = 12$ qubits and depth $d = 5$, we obtain a search space of size $3,609,673^5 = 6.13 \cdot 10^{32}$. However, it is clear that there is plenty of redundancy in this space when searching for a circuit that encodes a particular state, and we can also further reduce this space by excluding, for example, CX gates with control qubits that are certainly in the $|0\rangle$ state[37]. Yet, this shall serve as an indication that our search space grows rapidly with qubit size and depth; in this context, [27] states that the problem of optimal CNOT synthesis is likely not in the complexity class P. And while general asymptotic results are known in a monolithic setting [138], we are interested in finding efficient encoding circuits in a Type-II DQC setting for specific small instances.

4.1.2 Hardware Assumptions

Having identified Type-II systems as our architecture of interest, we next outline the hardware assumptions underlying `DQCode`. While these considerations explicitly impact the DQC simulations in Section 5, they also implicitly shape our optimisation.

- Motivated by modern ion trap architectures [5], we assume full intra-core connectivity for a Type-II architecture on the scale of the small qLDPC codes we are investigating in this work.
- Due to the relatively small number of cores in consideration, we consider full connectivity of QPUs via the quantum network. That means, both a quantum

[37] As will be discussed later, such reduction is implicitly used for the Genetic Algorithm by recombining valid CX circuits, and explicitly used for the Monte Carlo Tree Search by restricting the action space.

link (for distributed Bell state generation) and a classical link (for communication of measurement results) are generally available between all cores. In particular, every QPU hosts ((# of cores) − 1) communication qubits, such that at every timestep, a telegate can be performed between any two QPUs (mediated by the corresponding pair of communication qubits). Furthermore, we do not restrict the number of telegates that can be executed in parallel between different pairs of QPUs; nonetheless, we model accumulating idling times based on the availability of remote connections per timestep. Overall, the choice of a homogeneous fully connected network remedies any overhead of routing quantum links through a heterogeneous network [139], e.g., via quantum switches [140].

- We assume access to Z-basis measurements; to perform a measurement in another basis, the corresponding unitary transformation needs to be pre-applied to the state (e.g., a Hadamard transform to effectively measure in the X-basis).
- The overhead of compiling our optimisation gate set $\{H, CX\}$ to the gate set of a real quantum machine is assumed to be negligible. As an example, a CX gate only differs by single-qubit corrections from the Mølmer–Sørensen gate that is typically natively supported on ion trap devices [141, Section 3.1].

With these preliminaries in place, we next outline how we inform our optimisation with information about stabiliser interactions.

4.1.3 QEC-Aware Qubit Mapping via Hypergraph Partitioning

As discussed in Section 2.2, quantum compilation generally includes both a qubit mapping and a routing or scheduling component. In this work, we aim to alter the circuit structure to find efficient encoding circuits for a given DQC architecture. We do so by fixing one of the $\binom{n}{n_1,n_2,\dots,n_k}$ possible assignments of n qubits to k different cores of sizes $n_1, n_2, \dots, n_k$[38], similar to [26] when performing Quantum Circuit Discovery for monolithic architectures with restricted connectivity. Here, we motivate our qubit mapping by the QEC primitive of stabiliser measurements. In Section 2.1.11, we saw that fault-tolerant distributed stabiliser measurements are an alternative for state preparation, yet in general, such stabiliser measurements need to be performed frequently for the task of syndrome extraction. When using a non-local code on a Type-II architecture, it thus becomes a reasonable objective to minimise the telegate overhead associated with these repeated operations, by mapping qubits to cores in a way favourable for these measurements[39].

The idea to reduce remote operation overhead via mapping has been explored in literature before. Notably, [102] uses so-called *hypergraph partitioning* to assign qubits to QPUs. A hypergraph is a generalisation of a graph, where each hyperedge connects ≥ 2 nodes (compared to an edge connecting precisely 2 nodes in a usual graph). Denoting the set of hyperedges with E and the set of nodes (or vertices) with V, a *hypergraph* is the 2-tuple $H = (E, V)$, and a *k-way partition* of H is a partition $\Pi = \{V_1, \dots, V_k\}$ of its vertex set V. In the context of stabiliser measurements, we interpret each node as

[38] This count assumes indistinguishability of qubits within a core.

[39] As an aside, this QEC cycle aware mapping approach provides further justification for our choice of quantum gate teleportation over state teleportation, since qubits are mapped stationary to cores in anticipation of a recurring future operation.

a qubit, and each edge as one of the stabilisers, connecting the qubits that are part of the stabiliser measurement. An illustration of this can be found in Figure 7.

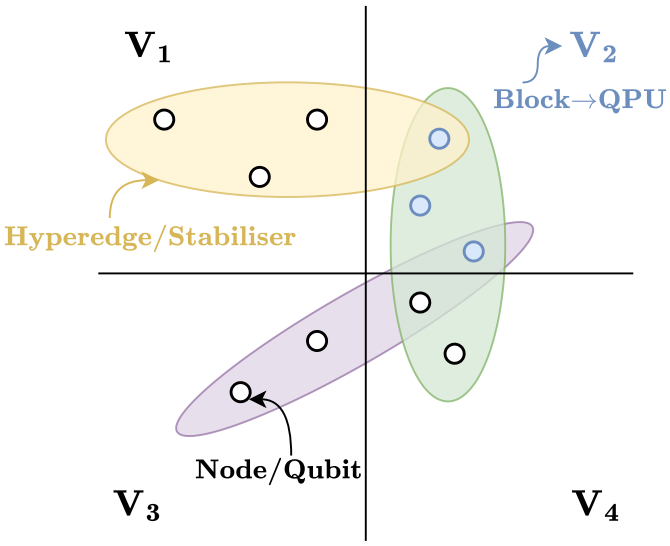


Figure 7: Illustration of a hypergraph, where nodes/qubits are connected by hyperedges/stabilisers. Each qubit belongs to one of the four blocks V_i of the partition $\Pi = \{V_1, V_2, V_3, V_4\}$, where each block is mapped onto the corresponding QPU. Adapted from [142].

The task of hypergraph partitioning is then to find a partition, i.e., an assignment of nodes (qubits) to disjoint blocks (QPUs), such that an objective function is minimised. Here, we use the *connectivity* metric

$$\mathfrak{c}_\lambda(\Pi) \equiv \sum_{e \in E} (\lambda(e) - 1), \tag{42}$$

where $\lambda(e)$ is the number of blocks that a given hyperedge $e \in E$ connects, depending on the partition Π (cf. [142]). For example, in Figure 7, the yellow and green hyperedges contribute $2-1=1$, whereas the purple hyperedge contributes $3-1=2$ to the sum. In the qubit mapping context, minimising $\mathfrak{c}(\Pi)$ over different qubit assignments (partitions) thus serves to minimise the telegate count associated with distributed stabiliser measurements. It should be noted that the objective function in Equation 42 does not count the actual number of e-bits required in a setting where every telegate is executed separately. However, it counts the number of stabilisers for which *at least one* telegate is required, which is still a useful proxy for the task of mapping qubits.

The *Karlsruhe Hypergraph Partitioning* (`KaHyPar`) framework [142] allows for a generalised version of hypergraph partitioning, where also vertex and hyperedge weights can be taken into account. In this work, however, we use `KaHyPar` as a heuristic solver on the unweighted hypergraph H defined above, since for the Bivariate Bicycle-$[[18,4,4]]$ and Trivariate Bicycle-$[[12,2,3]]$ code, all stabilisers have equal weight[40]. We then draw

[40] Also, all codes under consideration constitute rather small problem instances, such that we do not anticipate a weighted hypergraph to provide a significant improvement in mapping quality.

upon the tool's capability to find a balanced partitioning, where each block/QPU holds approximately the same number of vertices/qubits [41].

Below, we provide an example of how `KaHyPar` maps qubits onto Type-II DQC cores. In this context, we point out an important distinction between the stabiliser generators that we use for hypergraph partitioning, and the ones resulting from the canonical form of a QEC code, as discussed extensively in Section 2.1.6. While the canonical form is unique for some stabiliser group $\mathcal{S}$, and thus a fitting candidate to measure how close two states are[42], the bounded weight of stabilisers in the construction of Bivariate and Trivariate Bicycle codes is not necessarily maintained under canonicalisation. However, we deem it favourable for distributed stabiliser measurements to exploit precisely this boundedness (despite there being redundancy in the stabilisers, contrary to the canonical form); hence, the QEC cycle-aware hypergraph partitioning is provided with the low-weight stabilisers from the code constructions in Section 2.1.8. For the twelve qubits of the Trivariate Bicycle $[[12,2,3]]$ code, these stabilisers are

```
__XX__XX____
X___X__XX___
_X___XX_X___
X____X___XX_
_X_X______XX
__X_X____X_X
Z_Z____Z_Z__
ZZ______Z_Z_
_ZZ___Z____Z
___Z_ZZ___Z_
___ZZ__Z___Z
____ZZ__ZZ__,
```

where qubits are enumerated in increasing order from left to right. We can verify that there are twelve stabilisers (two of which are redundant to yield $12-10=2$ logical qubits), each of which has support on four qubits, and that each qubit is active in precisely four stabilisers[43]. Mapping the twelve data qubits onto two cores of size six each using `KaHyPar`, we obtain the assignment visualised in Figure 8[44].

4.1.4 Setting Up Fault Tolerance

Recall from Definition 6 that fault tolerance for CSS code encoding circuits requires us to construct a circuit in which X- and Z- errors do not propagate to higher-weight errors. The automated methods described in [27] allow us to find verification circuits that ensure this criterion for the CX circuits we are considering in this context. Since we may assume the gate set $\{H, CX\}$ for CSS code encoding circuits, analysing the error propagation of X- and Z-errors in Figure 3 is central to this task, and explicitly used in

[41] While hypergraph partitioning with common objective functions is a NP-hard problem [142], the discussion of the so-called *direct k-way partitioning* method employed for this task is beyond the scope of this work; the precise configuration of our hypergraph partitioning call can be inspected at `https://github.com/timneumann1/DQCode/blob/main/src/km1_kKaHyPar_sea20.ini`.

[42] This is captured by the concept of the tableau distance, as introduced in Section 4.2.2.

[43] Since both $A(x,y)$ and $B(x,y)$ in Example 7 are of weight 2, we obtain an upper bound of $2+2=4$.

[44] Appendix B.11 contains all qubit mappings obtained from hypergraph partitioning used in this study.

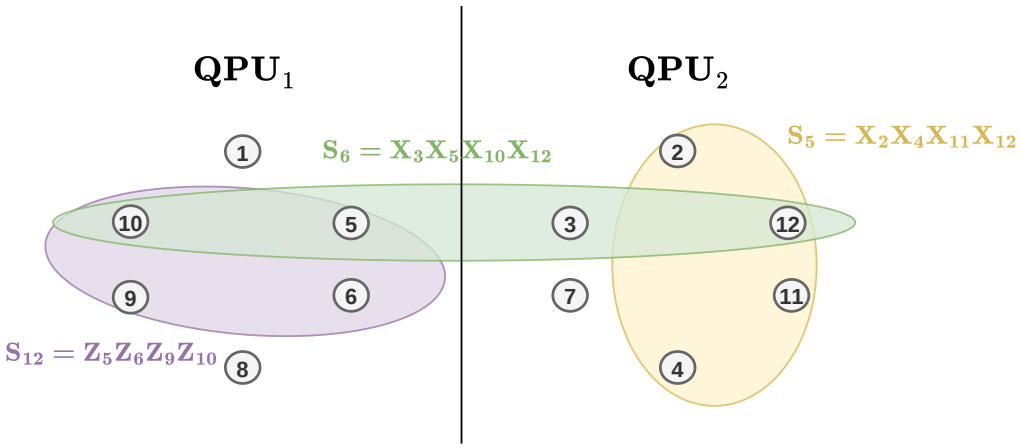


Figure 8: Mapping of the twelve data qubits of the Trivariate Bicycle-[[12,2,3]] code onto two QPUs, as a result of applying the hypergraph partitioning exposed by [142]. Three of the twelve stabilisers in 4.1.3 are visualised, two of which are local (of the twelve stabilisers, eight are non-local for the given mapping).

[27]. However, operating in a DQC system in this work, the telegate component in Figure 5 requires special attention. We adopt the following perspective: Errors can occur on the data qubits q_1 and q_2 as well as on the communication qubits that share the initial Bell state. But since the communication qubits are measured within the telegate protocol, and are thus effectively traced out of the system, we only need to investigate how errors spread onto the data qubits. Fortunately, in the entire telegate protocol, then, any single X- or Z-Pauli error propagates to at most one such error on the data qubits, as exemplified in Figure 9. In particular, errors on the data qubits propagate as usual, i.e., according to Figure 3, through the telegate circuit; any X-error on either of the communication qubits propagates to a X-error on q_2; and any Z-error on either of the communication qubits propagates to a Z-error on q_1. Note how the CSS code constraint is crucial here: If a Y-error occurs on the first communication qubit, it propagates to a Z-error on q_1 (via the CX gate) and an additional X-error on q_2 (via the Z-basis measurement), therefore abiding by Definition 6 only because X- and Z-errors can be corrected *separately*. Also, recalling that the distance of the CSS code is defined as the minimum over d_X and d_Z in Section 2.1.7, a valid verification circuit for a distance-d code separately flags both harmful X- and Z-errors, which will further be elaborated on in Section 5.1.

Even though the telegate protocol in Figure 5 maintains fault tolerance per Definition 6, the Bell state initialisation on the communication qubits is usually associated with elevated noise levels, as will be discussed in Section 5.2.1. Thus, the minimisation of telegates is per se a desirable optimisation objective. Within the introduction of DQC in Section 2.2, we already saw multiple compilation approaches that aim to reduce the number of telegates. In particular, some authors make explicit use of a generalised form of the protocol in Figure 5, where multiple telegates are performed using only one e-bit [109, 143, 113]. This technique relies on performing a grouping of gates into so-called *distributable packets* [109, Figure 1].

However, for many of these advanced compilation techniques, fault tolerance is not directly addressed: If a single e-bit mediates multiple telegates, then an error on

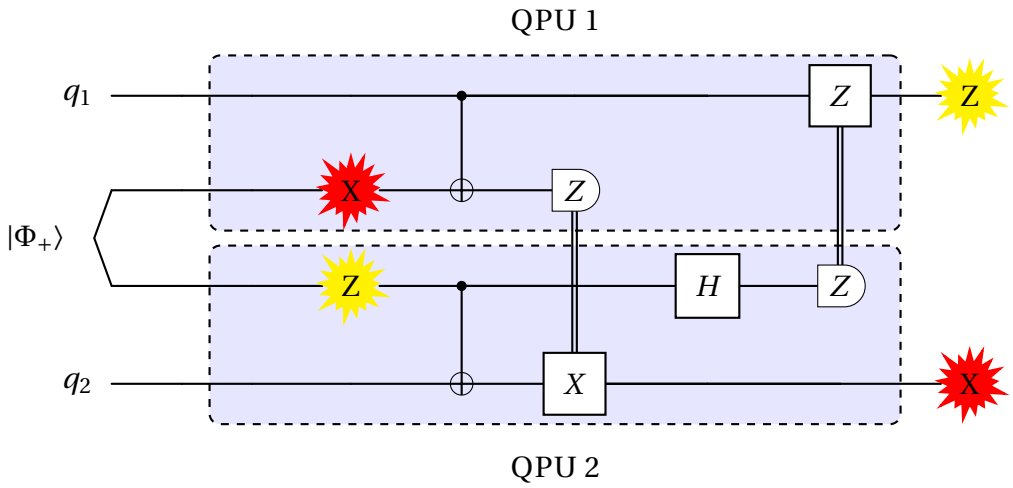


Figure 9: Error propagation in the telegate protocol: X- and Z -errors on either communication qubit propagate to a single X- or Z-error on the target qubit q_2 or control qubit q_1, respectively (simplified depiction, in which the X- and Z-errors at the end of the circuit are the propagated versions of the respective same-type error within the circuit). Adapted from [127] and [131, Figure S1].

a communication qubit can lead to multiple errors on the data qubits. For example, an X-error on the lower communication qubit in Figure 1 of [109] would lead to Z errors on both of the data qubits in core B. Such propagation cannot be detected by a straightforward adaption of the verification routine in [27]. As stated in [114], one way to address this issue is by "replac[ing] the state and gate teleportation primitives with their transversal, logical, equivalents". While this constitutes a solution for Type-III distributed architectures, it is not viable in the context of our Type-II study, where we explicitly distribute a code over multiple QPUs (in contrast to performing remote operations between separate code patches).

Therefore, to enable the addition of verification circuits to our optimised non-fault-tolerant encoding circuits, we opt to minimise telegates while adhering to the condition that every telegate requires a fresh e-bit. This choice of optimisation objective for the DQC setting does not guarantee improved logical initialisation error rates, but explicitly highlights fault tolerance as design principle at the optimisation level. Given this setup, we proceed to the description of the specific optimisation methods we use in `DQCode`

4.2 Optimisation Methods

In the following, we consider three different techniques to optimise state preparation circuits for CSS code logical $|0\rangle_L^{\otimes k}$ states on Type-II DQC architectures. Each optimisation technique aims at reducing the number of telegates, based on the pre-determined QEC cycle-aware mapping of qubits. While the Genetic Algorithm (GA) and Monte Carlo Tree Search (MCTS) inherently are circuit search methods, the DQC compilation method mainly serves as a preliminary step for the GA, and is presented first.

4.2.1 DQC-FT Compilation

In Section 2.1.11, we introduced Algorithm 1 that produces the baseline Gottesman encoding circuit for CSS codes. This algorithm does not incorporate any assumptions on the underlying connectivity of the quantum hardware. In order to reduce the gate count of such circuits in a DQC context, while not violating fault tolerance ('DQC-FT'), we adapt a method from [80] that exploits overlaps between CX gates. In particular, we replace certain sequences of four such two-qubit gates by three equivalent gates, effectively reducing the gate count by 1 for each sequence. For our study of (positive phase) CSS codes, Algorithm 2 captures a high-level version of these gate reductions, performed on the Gottesman encoding circuit from Algorithm 1.

Algorithm 2 DQC-FT Compilation of logical $|0_k\rangle_L$ Gottesman canonical circuit from Algorithm 1

Input: Gottesman encoding circuit for $|0\rangle_L^{\otimes k}$
Output: Preparation Circuit for $|0\rangle_L^{\otimes k}$ on n qubits with reduced (tele)gate count

```
compiled_circ = []                          ▷ Collects the final DQC-compiled circuit
remaining_gates = gottesman_circuit
singular_gates = []                         ▷ To be added to the end of compiled_circ
insertion_gate_qubits = []      ▷ Contains flagged qubits from compiled CX action
add H gates to compiled_circ
while !isempty(remaining_gates) do
    CX_{i→k} = remaining_gates[1]
    collect all CX_{j→k} gates that share target qubit k with CX_{i→k} in list CXs_{j→k}
    for CX_{j→k} in CXs_{j→k} do
        collect all CX_{i→m} gates that share control qubit i with CX_{i→k} in list CXs_{i→m}
        for CX_{i→m} in CXs_{i→m} do
            find first CX_{j→m} gate in remaining_gates
                        ▷ This yields the quartet {CX_{i→k}, CX_{j→k}, CX_{i→m}, CX_{j→m}}
            determine DQC-aware insertion gate (CX_{k→m} or CX_{m→k})
            if control of gate ∉ insertion_gate_qubits then
                add triplet to compiled_circ and remove quartet from remaining_gates
                        ▷ triplet depends on core assignment (cf. Appendix B.12)
                add qubits k and m to insertion_gate_qubits
                continue with while loop
            end if
        end for
    end for
    add CX_{i→k} to singular_gates and remove from remaining_gates
                                    ▷ in case no feasible quartet was found
end while
add singular_gates to compiled_circ
```

A supplementary explanation of the above compilation pass can be found in Appendix B.12. In Figure 10, we show the concrete example of the Gottesman encoding circuit (top) and its DQC-compiled counterpart (bottom) for the Trivariate Bicycle-[12,2,3] code on two cores of six qubits each, as depicted in Figure 8. Here, the DQC

compilation leads to a decrease of intra-core CX gates from twelve to eleven, and of telegates from nine to seven. The results of our DQC-FT compilation routine for all codes and architectures in this study may be inspected in Table 1.

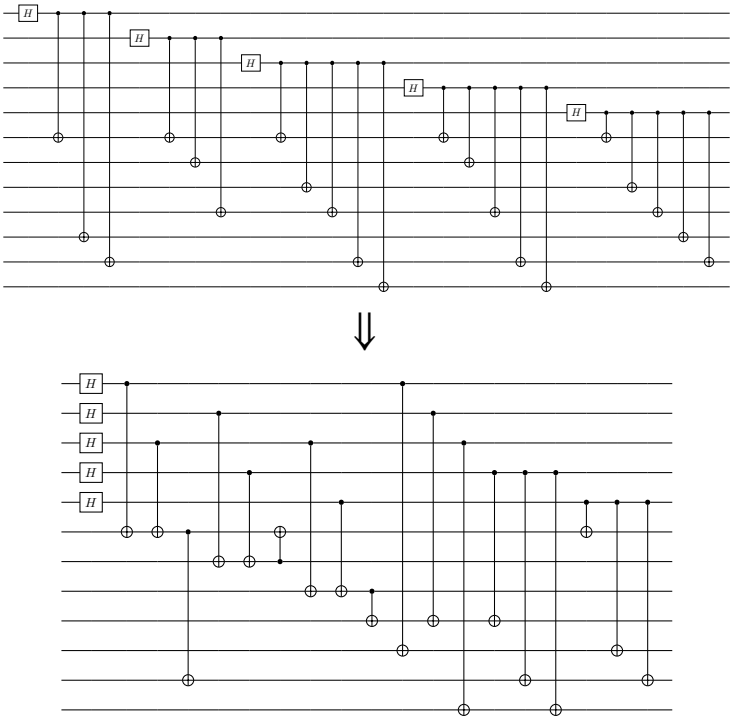


Figure 10: Comparison of $|00\rangle_L$ state preparation circuits for the Trivariate Bicycle-[[12,2,3]] code on two cores. Top: Gottesman encoding circuit from Algorithm 1. Bottom: DQC-compiled version thereof from Algorithm 2. Qubits listed in increasing order from top to bottom (q_1 on the top line, q_{12} on the bottom line), with mapping of qubits to cores as visualised in Figure 8.

4.2.2 Warm-Start Genetic Algorithm

While the DQC compilation pass already reduces the number of telegates, we next explore how to further optimise logical zero state encoding circuits. In the previous section, we started with a working encoding circuit and reduced its gate count via circuit equivalences. Here, we take the compiled circuit as initial solution and perform quantum circuit search, i.e., we search for a circuit that synthesises our desired target state via numerical optimisation. In particular, we firstly employ a *Genetic Algorithm* (GA). Such Genetic Algorithms are a class of gradient-free global optimisation algorithms that resemble an evolutionary process by evolving a *population* of *individuals* via multiple *generations* of *selection*, *crossover*, and *mutation* [144]. Genetic Algorithms tend to avoid local optima better than gradient-based optimisers, especially when the search space becomes very large, and have been utilised in a QEC context [145] as well as at the intersection of QEC and DQC [146, 118, 147]. The following paragraphs introduce the components of the GA, with the overall optimisation workflow being visualised in Figure 11.

Individuals and Generations. To enable our GA to find efficient circuits, we define each individual as a quantum circuit, which in turn is part of the larger population of

circuits. For our specific application case, these circuits consist of a fixed initial layer of Hadamard gates (which is not mutated throughout the search), followed by a variable number of CX layers[45]. For our experiments, we choose the number of individuals on the order of 10^4, and the number of generations on the order of 10^3. Naturally, we only encounter a small fraction of all possible circuits in the optimisation. To bias the search initially, we therefore utilise warm-start initialisation of the first population with the DQC-compiled circuits from Section 4.2.1.

Fitness. To gauge how well our circuit individuals are performing (and to enable a choice of parent individuals for crossover), we need to define some notion of *fitness* to navigate the search space. In order to evaluate how good a circuit individual performs in the task of efficiently synthesising the logical $|0\rangle^{\otimes k}$ state of some code, we generally want to take two aspects into account: (i) the fidelity of the circuit output state[46] with the target code state, and (ii) the circuit gate count. Regarding the fidelity term in an individual's fitness, an intuitive approach would be to determine $|\langle\psi_i|\psi_t\rangle|^2$, where $|\psi_i\rangle$ and $|\psi_t\rangle$ denote the output state of the circuit and the target logical zero state, respectively. Since we are working with Clifford circuits, we don't need to compute this fidelity with the full state vector, but can resort to more efficient methods of computing the inner product of two stabiliser states, as proposed in [30, Section III.] and improved in [34]. However, as noted in [26], such a metric does not provide a very dense reward to our GA algorithm. In particular, a circuit that comes 'close' to synthesising $|\psi_t\rangle$ (in the sense that it differs only by a few gates from a valid encoding circuit) might have comparably low fidelity. A good example of this is provided in [26, Figure 3], where the fidelity of the output circuit state does not grow monotonously when assessing a valid circuit gate-by-gate. To remedy this, [26] introduces the so-called *tableau distance*, which compares two stabiliser states based on the (dis)similarity of their respective tableaus. In this work, we utilise the Jaccard distance of two tableaus as distance metric, as described in Definition 7.

Definition 7: Jaccard Distance between tableaus

Given two binary tableaus S and T of equal size, the Jaccard distance $d_J(S,T)$ is defined as

$$d_J(S,T) := \frac{\left|\left\{i : S_i \neq T_i\right\}\right|}{\left|\left\{i : S_i = T_i = 0\right\}\right|}, \tag{43}$$

where i ranges over all entries in the tableaus. (cf. [26])

We use the tableau distance d_J to create the proxy $1.0 - d_J(T, T_{\text{target}})$ for the actual fidelity we are interested in. Note that this fidelity proxy

$$1 - d_J(S,T) = \frac{\left|\left\{i : S_i = T_i = 1\right\}\right|}{\left|\left\{i : S_i = T_i = 0\right\}\right|} \tag{44}$$

[45]The number of CX layers is not bounded directly, but implicitly via a minimal and maximal gate count for each individual. Due to this variability in depth, it suffices to store each circuit as a vector of operations, similar to [147].

[46]We apply all circuits on the physical all-zero state.

is analagous to the Jaccard index

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{45}$$

in statistics, which assesses the similarity of two sets A and B. In code, we adapt Definition 7 to the `QuantumClifford`-specific tableau, which encodes $\mathbb{1} \equiv 0, X \equiv 1, Y \equiv 2, Z \equiv 3$ as well as phases $+1 \equiv 0, -1 \equiv 2$. There, the denominator counts all nonzero tableau entries that agree, and the numerator counts all nonzero entries[47].

For the second component of our fitness function, we seek to penalise high (tele)gate count in a circuit individual. For this, we subtract a weighted gate penalty

$$P = P_1 \cdot (\#\text{ intra-core gates}) + P_2 \cdot (\#\text{ telegates}), \tag{46}$$

where $P_1, P_2 \in \mathbb{R}^+$ are chosen to penalise telegates more strongly than intra-core two-qubit gates. While precise weights need to be tuned depending on code and architecture, we generally choose P_2 one to two orders of magnitude larger than P_1 in this work.

Given these two components, we construct the fitness function

$$F = \alpha * \Big(1.0 - d_J(T, T_{\text{target}})\Big) - \beta(g)P, \tag{47}$$

where $\alpha \in \mathbb{R}^+$ is an additional hyperparameter, and $\beta(g) = 10^{-g/g_{\max}}$ is an exponential weighting term depending on the current generation $g \in \{0, 1, \ldots, g_{\max}\}$ and the overall number of generations $g_{\max} \in \mathbb{N}$. The latter term ensures that the relevance of the gate count penalty is relatively large in the beginning of a GA run (biasing towards efficient circuits), but loses one order of magnitude in strength towards the final generations (biasing towards high-fidelity circuits).

The presence of both fidelity and gate count terms in Equation 47 defines a trade-off between accuracy and efficiency of the encoding, which we can tune via the weighting hyperparameter α. For the given codes and architectures, we choose α one to two orders of magnitude larger than the telegate weight P_2. Figure 12 provides an exemplary evolution of maximal fitness, optimal gate counts and best fidelity per generation in the genetic search, in this case specific to the hyperparameters summarised in the accompanying table. We see that the algorithm starts with a population of fidelity-1.0 individuals from the DQC compilation phase, then trades off fidelity with reduced telegate count around generation 500, after which a fidelity of 1.0 is restored with lower gate count in the end. This behaviour is observed across most code-architecture pairs, and is a direct consequence of our generation-dependent penalty weighting in Equation 47.

Selection, Crossover and Mutation. Given some population and fitness function, we seek to define a rule that allows us to form a new *offspring* generation. For Genetic Algorithms, this is commonly captured in three steps: Selecting the best-fitness individuals from the parent generation (selection), recombining them (crossover), and changing individuals slightly (mutation). In this work, we perform selection tournaments of size 5, yielding the better-performing half of circuits. We then recombine these circuits

[47] This way, disagreement of Pauli operators on a qubit per stabiliser is only counted once in the numerator (instead of potentially twice, when treating the X- and Z-part of a tableau separately).

pairwise sequentially by randomly determining two splitting points within the CX bulk, and then crossing over the two parts, respectively. Lastly, each offspring circuit created this way is subjected to a random mutation, where a CX gate is altered (switching control and target, or replacing both) or deleted with a certain probability, or a CX gate is appended[48]. To complete the mutation step, each circuit is post-processed to remove unnecessary redundancies (such as repeated self-inverse CX gates), and ensure a given minimal and maximal size. The components of selection, crossover and mutation are visualised in Figure 11.

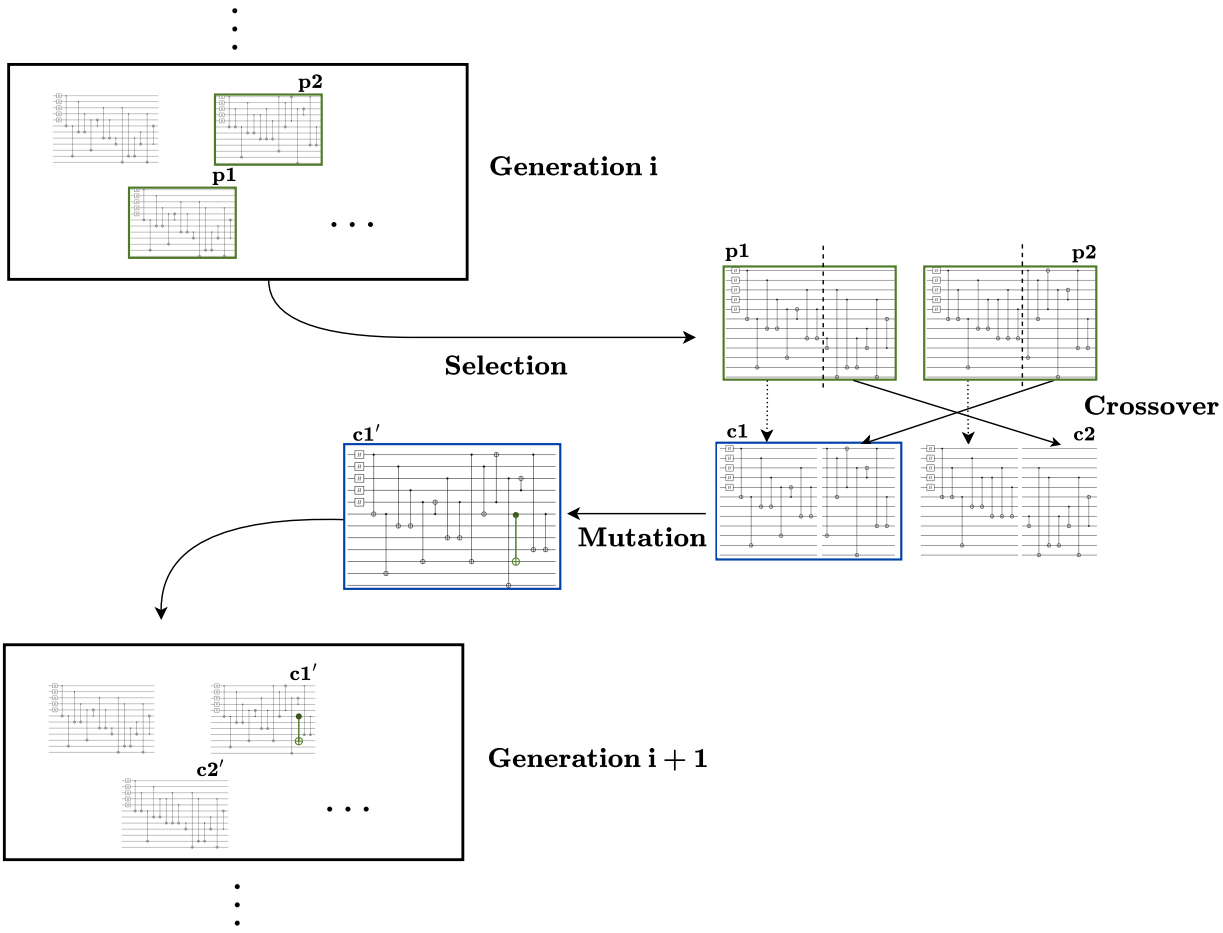


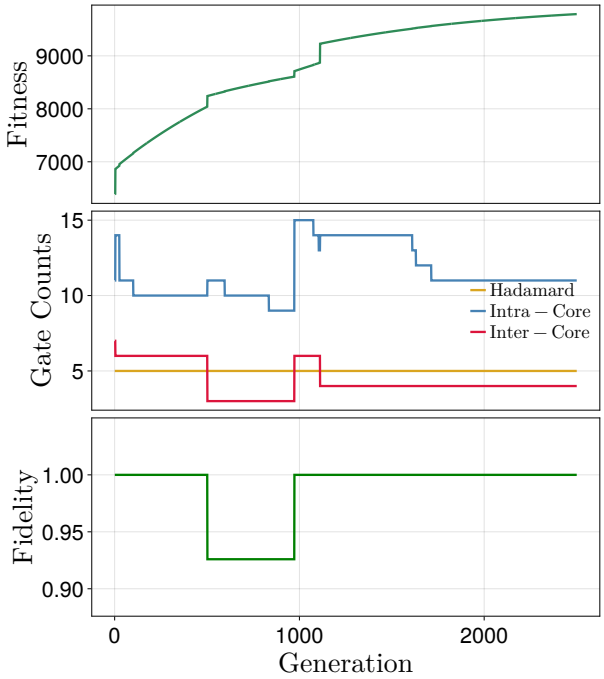
Figure 11: Exemplary selection, crossover and mutation operations for GA, where parent circuits p_1 and p_2 are selected from a given genereration i and re-combined to yield children circuits c_1 and c_2. The mutated individuals c_1' and c_2' are then added to generation $i+1$.

Given the tailored components of a dual fitness function, selection, crossover and mutation, we evolutionary optimise the CX encoding circuit within the paradigm of quantum circuit search, warm-started with the DQC compiled circuit individuals. In order to obtain a valid encoding circuit, we dynamically replace, per generation, the best-performing individual with respect to fitness, conditioned on the circuit achieving fidelity 1.0. Combined with maintaining a small number of elites, this ensures that

[48]The precise implementation of this can be found at https://github.com/timneumann1/DQCode/blob/main/src/genetic.jl#L406-443.

the final valid best-performing circuit is also the optimal circuit throughout the entire search with respect to gate count. The state preparation circuit for the Trivariate Bicycle-[[12,2,3]] code on two cores is presented in Figure 12, as extracted from a genetic search using the accompanying hyperparameters. This circuit only needs four telegates, compared to nine and seven for the Gottesman encoding circuit and its DQC-compiled version, respectively. General results of the GA across codes and Type-II architectures are visualised in Figure 15, as part of a larger comparison of optimisation methods. Appendix B.13 collects the state preparation circuits discovered by the Genetic Algorithm for all QEC codes in this work.

Hyperparameter	Value
No. of Individuals	25,000
No. of Generations	2,500
Max. Gate Count	100
Mutation Rate	0.85
Tournament Size	5
α	10^4
P_1	10
P_2	$5 \cdot 10^2$



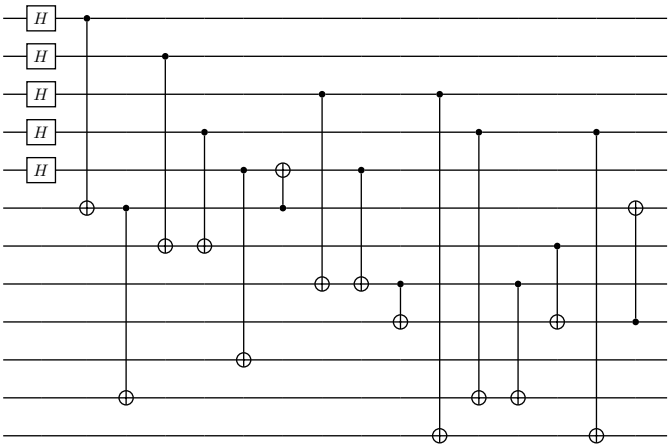


Figure 12: Logical $|00\rangle_L$ state preparation circuit search for Trivariate Bicycle [[12,2,3]] code on two cores via Genetic Algorithm. Top Left: Hyperparameters for GA search, notation partly deriving from Equation 47. Top Right: Evolution of GA metrics over generations. Bottom: Optimised state preparation circuit.

4.2.3 Monte Carlo Tree Search

As an alternative to performing a warm-start GA, we investigate a method that builds encoding circuits gate by gate from scratch. For a thorough introduction to *Monte Carlo Tree Search* (MCTS), including the mathematical formalism, we refer the interested reader to Appendix B.14. Here, we restrict our attention to presenting a concise motivation as well as a description of the main algorithmic steps. Also, Appendix B.15 adds some remarks relevant to the inner working of the MCTS, which are however not strictly needed to understand the general mechanism behind this search algorithm.

Monte Carlo Tree Search is a tree-based optimisation technique that lets us identify the optimal next action to take, given a certain state of the environment. This heuristic search algorithm played an integral role in the famous *AlphaGo* algorithm [148], and has also been featured in the context of finding encoding circuits for small error-detecting codes [149]. In our context, actions are quantum gates, states are quantum circuits (or the corresponding stabiliser states resulting from the noiseless circuit being executed), and a combination of fidelity and gate counts (cf. Equation 48) defines the reward used to assess how 'good' an action is. In particular, we attribute a so-called *Q-value* to each state-action pair, measuring the quality of this action. The Q-value is the sum of (i) the immediate reward from performing the action (applying a gate given a quantum state), and (ii) the (discounted) value of continuing to build a circuit with the chosen next gate. Maximising over the Q-values of the actions allows us to perform a search that looks ahead, at the cost of having to compute many trajectories when determining the future value of choosing some action. Especially when the state space is large, this becomes intractable, which is why MCTS is used to effectively approximate the true Q-value of performing an action via repeated Monte Carlo sampling.

In particular, the MCTS sampling serves to build a search tree of states and actions. An example of such a tree is displayed in Figure 13, where state nodes are marked in blue and action nodes are marked in red to green hue (depending on the frequency of them being sampled). Since we are working with a deterministic MDP, any state-action node uniquely maps to a new state. Even though the example depicted in the figure is merely for illustration purposes[49], we can see that each action node holds its corresponding Q-value estimate. For each layer of the tree, we may then select the action with the largest Q-value to be appended to the given circuit, building it gate by gate.

There are four phases to a Monte Carlo Tree Search, as visualised in [149, Figure 4]. For each iteration of the search, i.e., for each new action to be performed given the current state (i.e., gate to be added to the current circuit), these four phases are performed sequentially. This builds the search tree akin to Figure 13 up to the computational budget set for the sampling.

- **Selection:** Firstly, we start at the root node (corresponding to an empty circuit) and traverse a given tree until we reach a state node that is not fully expanded, i.e., missing child nodes corresponding to some actions (gates).
- **Expansion:** Secondly, we expand this state node s with an action node a, and thus obtain the corresponding next-state node s'.

[49] For example, we only consider the Hadamard layer, and the search depth is set to 1, i.e., only the next layer is considered in the optimisation at each step.

- **Simulation:** Thirdly, we simulate the decision process from root to sampled leaf s'. Framing the MCTS as a circuit search problem, this means that we add the gates corresponding to the state nodes encountered on this trajectory to a quantum circuit, and simulate this circuit. This yields a state tableau T, from which we can can extract our fidelity proxy via the tableau distance $1.0 - J(T, T_{\text{target}})$, as described in Section 4.2.2. The reward of the leaf state $R(s, a, s')$ is then calculated in analogy to the GA fitness via Equation 47. However, for this sequential decision making process, we consider a marginal improvement, where we don't measure the absolute tableau distance and gate count of a new state (circuit) s', but rather its change compared to the fidelity and gate count for circuit s. In particular, for fidelity proxies $\hat{F}_s \equiv 1.0 - d_J(T_s, T_{\text{target}})$ and $\hat{F}_{s'} \equiv 1.0 - d_J(T_{s'}, T_{\text{target}})$, we define

$$R(s, a, s') = \gamma(\hat{F}_{s'} - \hat{F}_s) - P_1 \cdot \mathbb{1}_{\text{a(s) is telegate}} - P_2 \cdot \mathbb{1}_{\text{a(s) is intra-core gate}}, \tag{48}$$

  where $\gamma, P_1, P_2 \in \mathbb{R}^+$ are hyperparameters and $\mathbb{1}$ denotes an indicator function.

- **Backpropagation:** Lastly, equipped with this new reward, we traverse the tree back from leaf to root, and update the Q-value of all action nodes as well as the visit counts to all nodes.

The `julia` implementation of the algorithm[50] is based on the `MCTS` sub-library of the `JuliaPOMDP` library [150]. As alluded to earlier, Appendix B.15 contains some further insights and remarks regarding the algorithm and its implementation. To obtain the results reported in Figure 15, we performed extensive hyperparameter tuning, details of which can also be found in the appendix.

Figure 14 displays the result of performing a MCTS for circuit synthesis of the logical $|00\rangle_L$ state of the Trivariate Bicycle code on two cores, as depicted in Figure 8, including the accompanying hyperparameters. The gates visible in this circuit are the respective optimal actions at each step. The MCTS yields an encoding circuit with four telegates, matching the Genetic Algorithm in this metric, yet exhibiting more (fifteen) intra-core gates than the GA.

[50] We refer the reader to `https://github.com/timneumann1/DQCode/blob/main/src/mcts.jl` for our MCTS implementation.

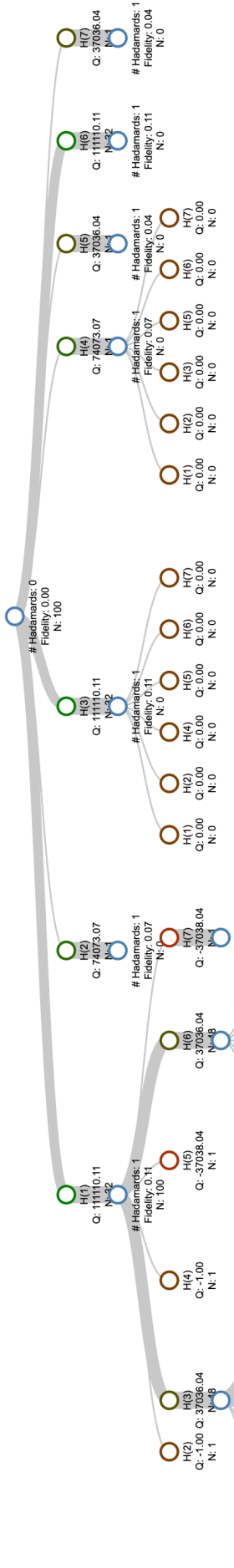
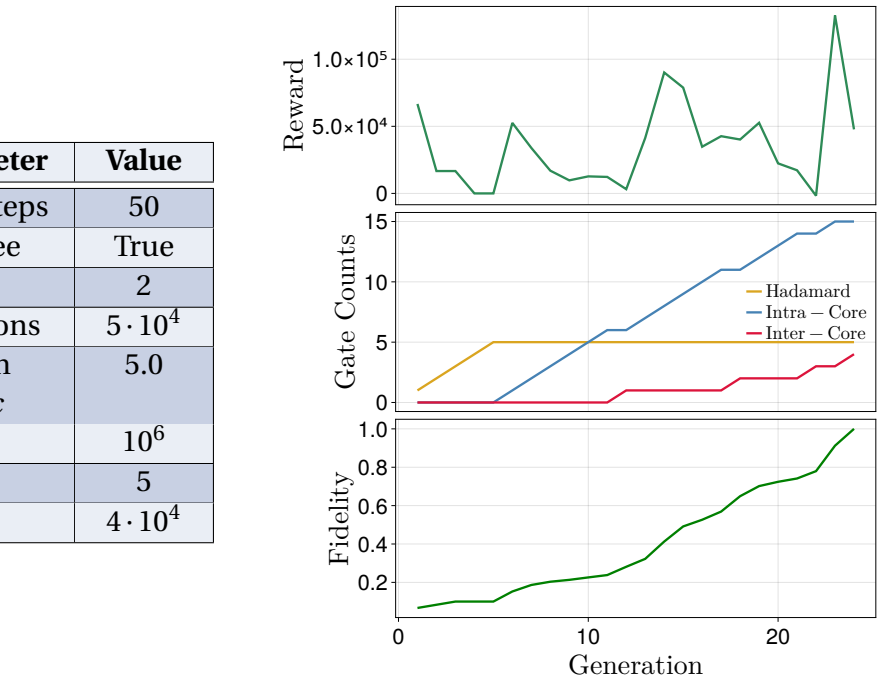


Figure 13: Example tree produced via Monte Carlo Tree Search. State nodes are blue, action nodes are red to green depending on the number of times they have been traversed in the sampling process. Based on the Q-values, the extracted Hadamard gates from this tree are H_1 (first layer), H_3 (second layer) and H_6 (third layer), where ties are broken in favour of lower qubit index.

Hyperparameter	Value
Max. No. of Steps	50
Reuse of Tree	True
Depth	2
No. of Iterations	$5 \cdot 10^4$
Exploration Constant c	5.0
γ	10^6
P_1	5
P_2	$4 \cdot 10^4$



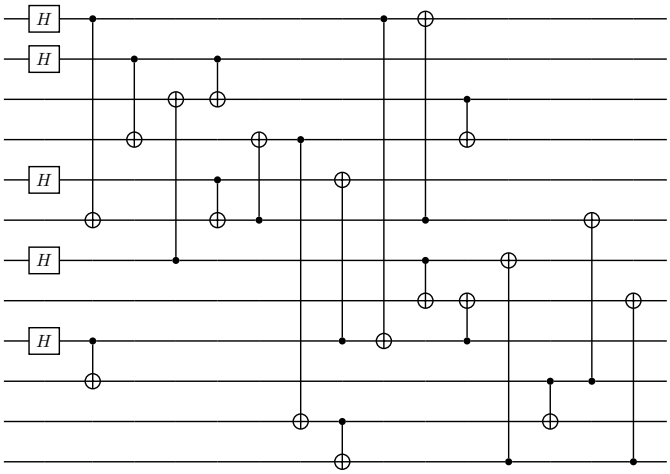


Figure 14: Logical $|00\rangle_L$ state preparation circuit search for Trivariate Bicycle-[[12,2,3]] code on two cores via Monte Carlo Tree Search. Top Left: Hyperparameters for MCTS, notation deriving from Equation 48 and Appendix B.15. Top Right: Evolution of MCTS metrics over generations. Bottom: Optimised state preparation circuit.

4.2.4 Baseline Methods

In order to compare our GA and MCTS to existing state preparation approaches, we draw upon four methods for state preparation: (i) the Gottesman encoding circuit from Section 2.1.11, (ii) a standard `Qiskit` [151] stabiliser state synthesis formalism [32], (iii) the *Munich Quantum Toolkit* (`MQT`) `QECC` tool [152], and (iv) a DQC compilation based on telegate grouping. For the second and third baseline, we refer to the cited works for a comprehensive introduction. We evaluate the DQC gate counts of the extracted circuits by counting the number of telegates that would be required to implement them, based on the mapping strategy defined in Section 4.1.3.

- `Gottesman`. Firstly, we determine the gate count required for the standard Gottesman encoding circuit defined in Algorithm 1.

- `Qiskit Greedy`. Secondly, `Qiskit` exposes a greedy Clifford compiler outlined in [32, Appendix A][51]. Since this method may introduce SWAP gates to the circuit, we compile each SWAP gate via the equivalence

$$SWAP_{i,j} = CX_{i\to j}CX_{j\to i}CX_{i\to j}. \tag{49}$$

- `MQT-QECC`. Thirdly, the `MQT` tool comprises (i) an optimal method based on satisfiability solving (SAT) to construct gate-optimal state preparation circuits, and (ii) a heuristic method in which a given tableau is reduced to a product state via Gaussian elimination guided by cost functions, such that the encoding circuit can be extracted from the reduction operations in reverse order [27][52]. Since the first method allows us to find gate-optimal circuits[53] in a monolithic setting, we synthesise circuits using both the SAT-based and the heuristic solver, and report the gate count for the circuit requiring fewer telegates based on our Type-II mapping.

- `E-Bit Compiled`. Whereas the above methods target encoding circuits for monolithic architectures (and without connectivity constraints), we lastly perform a DQC-compilation for comparison[54], in which multiple telegates are grouped for distribution via a single e-bit, as described in [102, 113]. This compilation is applied to the standard Gottesman encoding circuit from Algorithm 1, whose encoding structure allows for a straightforward grouping. While instructive as baseline, it should be noted that such extended quantum gate teleportation violates our fault tolerance design principle, resting on the assumption that each telegate consumes one e-bit, as discussed in Section 4.1.4. When reporting the compilation results in Table 1, we count one telegate per gate group, since multiple inter-core gates are performed with a single e-bit [55].

Apart from these methods, we omit a comprehensive comparison with methods that additionally incorporate quantum state teleportation for improved routing of a circuit, such as `TeleSABRE` [103, 153]; these compilation methods also accept some circuit and define an advanced routing or scheduling procedure on them, and are thus quite similar in spirit to the DQC-grouping approach. Nonetheless, we illustrate the potential effect of including such capability on the optimisation results for the example of the $[[12,2,3]]$ code on three cores.

[51] The code implementation can be inspected at https://github.com/Qiskit/qiskit/blob/main/qiskit/synthesis/clifford/clifford_decompose_greedy.py (accessed 15.07.2026).

[52] Please refer to https://github.com/munich-quantum-toolkit/qecc/tree/main/src/mqt/qecc/circuit_synthesis (accessed 15.07.2026) for an implementation of these tools; note however that this software package is under active development, so in this work, we draw upon Release 1.9.0, as exposed in the fork at https://github.com/timneumann1/qecc.

[53] The `MQT-QECC` software offers methods to find efficient encoding circuits with respect to gate count and depth, where we utilise the former.

[54] Overall, the literature for distributed quantum circuit search is sparse; in order to still allow for a comparison with DQC-tailored methods, we thus resort to the DQC compilation of a given circuit.

[55] For the other methods, telegates and e-bits are counted interchangeably.

4.3 Results & Discussion

The results of our GA and MCTS optimisation runs as well as the baseline comparisons are summarised in Table 1. Each cell comprises both the intra-core and inter-core (telegate) counts of the encoding circuits found by the respective method. Note that the number of single qubit gates is constant for all circuits in a row, hence we omit it here. The QPU sizes reported in the table concern the data qubits alone; for our DQC simulations in Section 5, we will append each core with the required communication and ancilla qubits. Each optimised circuit reported in the table prepares the logical $|0\rangle_L^{\otimes k}$ state of the corresponding code, which is described by appending the Z-type logical operator(s) to the stabiliser group (cf. Appendix B.11). The only exception to this is the Shor code, where the choice of logical Z-operator corresponds to the logical $|+\rangle_L$ state [13, 26] — this is however merely a matter of labeling.

QEC Code	QPU Sizes	Qiskit Greedy [32]	MQT-QECC [152]	Gottesman [17]	E-Bit Compiled	DQC-FT Compiled [4.2.1]	Warm-Start GA [4.2.2]	MCTS [4.2.3]
[[7,1,3]] [14]	[4,3]	7|9	**2**|6	4|5	**2**|7	3|5	**2**|6	4|5
[[9,1,3]] [13]	[3,3,3]	10|6	**2**|6	6|4	**2**|8	4|5	**2**|6	**2**|6
[[12,2,3]] (2.1.8)	[3,3,3,3]	12|9	9|7	15|6	7|14	11|7	**6**|8	10|7
	[4,4,4]	11|10	10|6	13|8	8|13	10|8	**5**|10	9|9
	[6,6]	10|11	5|11	9|12	5|16	7|11	**4**|11	**4**|15
[[17,1,5]] [154]	[9,8]	28|21	4|21	10|22	**3**|29	8|20	**3**|23	—
[[18,4,4]] (2.1.8)	[3,3,3,3,3,3]	56|5	28|4	42|5	29|18	35|5	**19**|15	32|16
	[6,6,6]	43|18	21|11	35|12	13|34	28|12	**9**|22	29|16
	[9,9]	36|25	17|15	27|20	**7**|40	21|19	8 |28	19|26

Table 1: Type-II DQC gate counts for $|0\rangle_L^{\otimes k}$ encoding circuits across different QEC codes and architectures (rows) and optimisation techniques (columns). Each cell reports the number of telegates and intra-core gates (in this order), evaluated on the QEC cycle-aware DQC mapping derived from hypergraph partitioning of qubits, as described in Section 4.1.3. The best-performing telegate count per row is highlighted.

It shall be noted that a typical GA or MCTS run takes on the order of a few hours on a MacBook Air (Apple M3, 8-core) for the final sets of hyperparameters[56], whereas the gate-optimal method of the `MQT-QECC` baseline was allotted a computational budget

[56]The hyperparameters used in our experiments can be inspected at https://github.com/timneumann1/DQCode/blob/main/src/experiment_config.jl. For the optimisation runs across different QPU sizes for a fixed code, we only vary a small number of weighting parameters without changing the hyperparameters driving runtime (e.g., the number of generations and individuals for the GA, or the number of samples for the MCTS) too much.

of approximately nine hours[57]. All other methods terminate within seconds, where a larger computational budget would not increase result quality. While the runtime for GA and MCTS might seem extensive, we stress that such circuit search only needs to be performed a fixed number of times, until a circuit of sufficient quality is found. This circuit can then be used indefinitely for quantum experiments in a production or experimental lab setting. Assuming that classical compute is cheap and abundant compared to quantum compute, this can be an acceptable strategy to reduce costly quantum operations.

In Table 1, we see that the Genetic Algorithm, warm-started with the DQC-compiled Gottesman encoding from Section 4.2.2, outperforms the other methods among almost all codes and architectures. In particular, compared to the `E-Bit Compiled` method as next best-performing approach, it achieves an average reduction of telegate count by 23.9% across the Trivariate Bicycle code experiments, and matches the count for the Steane, Shor and Color code, while requiring less intra-core gate for all code-architecture pairs in question. For the Bivariate Bicycle code, we significantly reduce the telegate count for the highly modular architectures (by 10 and 4 telegates for the system of six and three cores, respectively), while requiring one additional telegate for the two-core system. Compared to the naïve Gottesman encoding circuit, the GA reduces the telegate count by an average of 59% for the Trivariate Bicycle $[[12,2,3]]$ code, and by an average of 66.5% for the Bivariate Bicycle $[[18,4,4]]$ code experiments, respectively. Furthermore, we see that the GA method typically trades off lower telegate counts with a slightly elevated intra-core CX count, compared to the baseline methods, which is a consequence of the fitness function weights discussed in Section 4.2.2.

Consequently, we can state that our GA method yields a significant decrease in telegate count compared to the baselines in almost all cases, and provides viable encoding circuits with lowered communication overhead in the specific Type-II DQC setting. This is not entirely surprising for the `Qiskit Greedy`, `MQT-QECC` and `Gottesman` methods[58], which are inherently monolithic baseline methods and thus not optimised for a DQC setting. However, the fact that the GA circuit search also performs similar to a DQC-compilation method, which allows for telegate grouping, exemplifies that the fault tolerance design principle does not hinder our optimisation effort.

Since the telegate count is the most relevant metric here, we visualise it separately for the Bivariate and Trivariate Bicycle codes in Figure 15, omitting the `E-Bit Compiled` method, since it is not abiding by our fault tolerance principle. Generally, we find that the number of remote gates required for encoding the Multivariate Bicycle codes can be decreased further for less distributed system. This is reasonable, since for a smaller number of cores, more qubits interact via intra-core gates.

Interestingly, incorporating quantum state teleportation as additional compilation tool does not yield better results when applied to the case of the Trivariate Bicycle code on three cores. Here, we use the open-source implementation of `TeleSABRE` [103] to compile the Gottesman and warm-start GA circuits, which requires 13 state and two gate teleportations for the former, and five state and two gate teleportations for the

[57] Details of the timeout implementation may be inspected at https://github.com/timneumann1/qecc/blob/main/scripts/state_encoding.py#L25.

[58] It is noteworthy that the inherently monolithic `MQT-QECC` matches our DQC-optimised GA method in telegate count for the Steane and Shor code, while outperforming the `Gottesman` method by 43.8% in telegate count across all codes and architectures, on average. While it is sensible that a reduction of the gate count in an unconstrained monolithic setting also benefits the DQC telegate count to some extent, this highlights the strength of the tool.

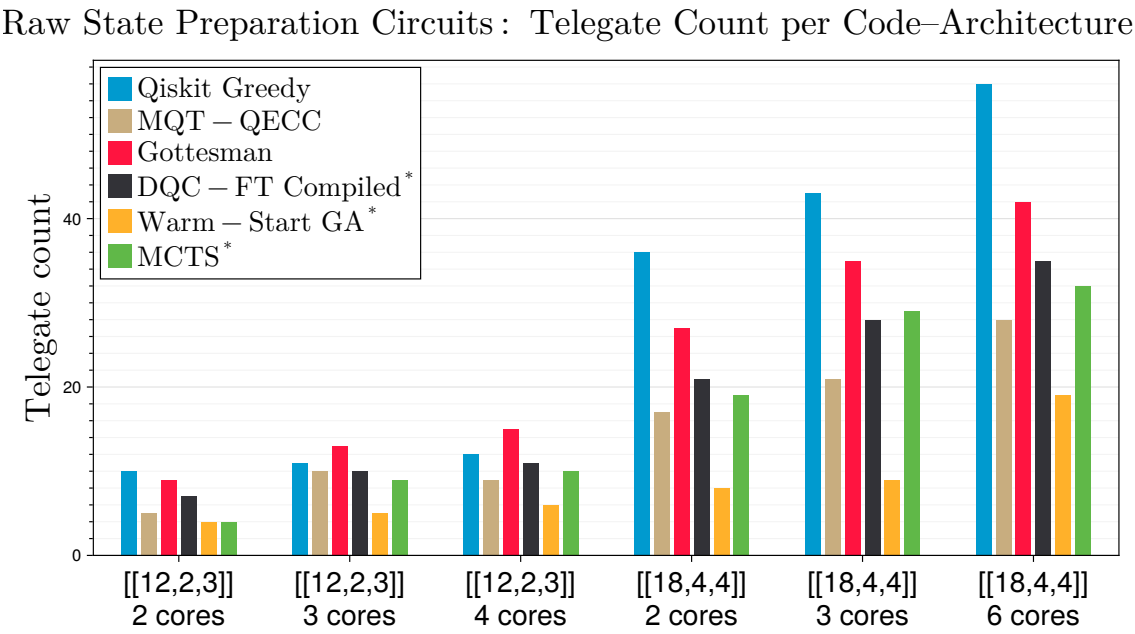


Figure 15: Telegate count comparison for state preparation circuits of Bivariate and Trivariate Bicycle codes on different Type-II architectures. Methods devised as part of `DQCode` are marked with an asterisk.

latter[59]. Given our Type-II assumptions, this therefore does not yield a decrease in the required remote-entanglement resources compared to the five telegates reported in Table 1, since each quantum state teleportation also consumes one e-bit.

The Monte Carlo Tree Search is generally outperformed by our GA approach, especially for larger codes; for the Color-$[[17,1,5]]$ code, the MCTS fails to find a circuit altogether. Still, it is notable that the MCTS matches the best telegate count across all methods for the Shor code and the Trivariate Bicycle code on two cores, despite constructing the circuit from scratch.

It is interesting that our DQC-optimised state preparation quantum circuit search discovers efficient encoding circuits for a monolithic setting as a byproduct. For example, we find a circuit with eight two-qubit gates for the Steane-$[[7,1,3]]$ code (cf. Table 1), which matches the gate-optimal method from [27] as well as the (monolithic) overlap method [80, Figure 5] and the result of the RL approach discussed in [26]. Similarly, our approach matches the results for the Shor code in [27, 26] in preparing the logical $|0\rangle_L$ state[60], and outperforms the heuristic state preparation method exposed in `MQT-QECC` in terms of overall two-qubit gate count across all Trivariate Bicycle code experiments.

With respect to our first research question from Section 3.2, we have validated the warm-start GA as an effective method yielding significantly reduced telegate counts across the code-architecture pairs in question. As a consequence of these results, and since the discovered circuits abide by our fault tolerance principle, we focus the DQC simulation in Section 5 exclusively on the circuits obtained from the GA.

[59] The standard configuration reported at `https://github.com/Haimrich/telesabre` (accessed 17.06.2026) was used for this task.

[60] With $|0\rangle_L$ matching the notation in [27, Table I], while in [26], for example, this state is referred to as $|+\rangle_L$.

4.4 Limitations & Future Work

We would be remiss not to point out the limitations of the methods discussed in this section, some of which motivate possible avenues for future work. Firstly, our methods are specifically tailored to the encoding of a single state, in this case the logical $|0\rangle_L^{\otimes k}$ state. We discussed in Section 3.1 that encoding of a general quantum state is not possible in a fault-tolerant manner with the methods used in this work. Also, optimisation routines for more general encoding circuits (even for the encoding of, say, all computational basis states), would add many more terms to the fitness function, drastically increasing complexity. Since the logical zero state provides a useful starting point for subsequent logical operations, the resulting loss of generality is an acceptable trade-off for a more tractable optimisation problem. As a direct extension of our approach, our focus on qLDPC-CSS codes[61] could be lifted by considering a more general Clifford gate set, e.g., including the phase gate. In particular, we assume the overhead of compiling our optimisation gate set $\{H, CX\}$ to the gate set of a real quantum machine to be negligible for the optimisation task at hand. And in terms of the networking assumptions in Section 4.1.2, future work could explore heterogeneous networks or limited intra-core connectivity (cf. [26]), necessitating advanced routing and scheduling of remote entanglement.

In order to improve the existing methods, a couple of immediate actionables shall be mentioned. Firstly, the DQC-FT compilation method only performs one greedy compilation pass, which could be improved by taking into account how reducing a certain quartet impacts further gate combination opportunities. Secondly, the MCTS method implemented here is a vanilla version, and could be extended to more advanced frameworks such as Progressive Widening [155]. Lastly, the GA fixes a placement of Hadamards and is warm-started with an entirely homogeneous initial population. Even though this algorithm performs well for the small codes in our study, especially the diversity in population should be increased in future work. One possible approach here would be to introduce an additional measure that incentivises diversity at the cost of short-term fidelity. Across all three approaches, it could further be worthwhile to consider quantum state teleportation as DQC primitive. And besides these methodological enhancements, our methods would benefit from additional hyperparameter tuning[62].

Even with such advances, the most severe limitation of the methods presented here is computational cost, which restricts their scalability. While the DQC-compilation of the Gottesman circuit typically terminates within a second, the GA and MCTS exhibit run times on the order of hours, and therein already rely on extensive hyperparameter tuning[63]. And in all experiments investigating larger problem sizes than the ones reported in Table 1, neither of the optimisers succeeded in finding valid (in the case of the MCTS) or improved (in the case of the warm-started GA) encoding circuits. While diffi-

[61] When considering codes other than qLDPC codes, it would be desirable to refine the hypergraph partitioning, since stabiliser weights might vary significantly.

[62] In this work, hyperparameter tuning is informed by empirical observations such as the expected fidelity improvement associated with adding a telegate, and thus highly manual. In this context, it is difficult to transfer insights from one such pair to another; something that RL, for example, might do better under the paradigm of *transfer learning* [26].

[63] Even for the small codes reported here, balancing gate-count penalties and fidelity improvements by tuning the fitness (or reward) weights in Equations 47 and 48 in an exploration-exploitation trade-off is a challenging task.

culties in scalability are to some extent expected, since GA and MCTS are circuit search (and not compilation) methods, this raises the question of whether we are facing an issue of insufficient hyperparameter tuning, or a structural limit of these optimisers.

Scalability concerns are a recurring theme in quantum circuit search, due to search spaces increasing exponentially with qubit count. As such, also [26] and [27] work with small problem instances. Nonetheless, the heuristic search exposed in [27] (cf. Section 4.2.4) generally allows circuit search for larger instances, and the authors further extend their approach with a rollout method in [156, Section III.C.], yielding good results up to a 144-qubit code[64]. Hence, as an alternative to our scalable (but less performative) DQC compilation technique, it would be interesting to extend the heuristic synthesis methods in [156] to incorporate a DQC component[65]. For the near-term implementations we seek to enable with our tailored Type-II encoding circuits, however, it is acceptable to restrict to smaller code sizes.

Finally, both the telegate minimisation and fault tolerance design principle need to be scrutinised under an important limitation: Although they yield circuits with reduced inter-core communication overhead that adhere to fault tolerance, these principles do not necessarily guarantee optimal performance in an actual DQC computation. Firstly, the telegate count is only a proxy for performance in a noisy (DQC) setting, since even circuits with relatively low such count might have large depth (and thus accumulate noise) due to scheduling conflicts in the face of limited communication qubits (cf. Section 5.2.2). However, while it is in general difficult to optimise for gate count and depth concurrently, we expect the effect of reduced depth to be small compared to the benefit of reduced telegate counts. And secondly, even though this violates fault-tolerance, it might be advantageous to execute multiple remote gates via one e-bit, given that the resulting reduction in telegate count or circuit depth outweighs the additional error propagation imposed by the loss of fault tolerance. Here we may argue that our warm-start GA performs on par with a gate grouping compilation method (cf. Table 1), such that this effect can be expected to be negligible for the small codes at hand. Nonetheless, the possibility of circuits obtained from different design principles (such as depth optimisation or non-fault-tolerant circuits) outperforming our approach in terms of simulated logical initialisation error rates (cf. Section 5.2.4) remains.

To assess whether the optimisation and design principles employed in this section translate into good performance under realistic hardware constraints, in the following section we simulate the discovered state preparation circuits in a noisy DQC setting.

[64] A comparison with the new methods devised in [156] was not feasible since the publication of the paper coincided with the termination of the code development phase of the present work.

[65] In particular, information about core connectivity could guide the greedy search therein.

5 DQC Simulation of Fault-Tolerant Encoding Circuits

> "When the elementary error rate is less than the logical error rate, coding is not helpful. When the logical error rate is less than the elementary error rate, coding is helpful. The pseudo-threshold thus captures the crossover point."
> – Barbara Terhal, [6]

In the previous section, we elaborated on methods yielding DQC-tailored state preparation circuits. Next, we seek to make these $|0\rangle_L^{\otimes k}$ synthesis circuits fault-tolerant, and assess their performance in a realistic Type-II setting. For this, we draw upon verification circuits, flagging harmful errors, and perform a DQC simulation of the resulting circuits given a particular noise model. From this, we extract a set of metrics that help us to numerically assess the quality of the circuits with respect to error rate and fault tolerance. As before, we primarily focus on the Trivariate Bicycle-$[[12,2,3]]$ and Bivariate Bicycle-$[[18,4,4]]$ codes here[66]. Firstly, a discussion of the verification circuit approach is provided in Section 5.1, followed by a treatment of the simulation methods in Section 5.2. We then report our results in Section 5.3, and close with an outline of the limitations of this study as well as potential future lines of work in Section 5.4.

5.1 Verification Circuits For Fault-Tolerant State Preparation

In order to simulate fault-tolerant state preparation circuits in a DQC setting, we need to synthesise verification circuits based on the optimised circuits from Section 4, as alluded to in Section 2.1.11. In this section, we discuss the need for such verification gadgets in more detail, outline their construction, and argue why we can adapt them to our Type-II DQC setting.

Harmful Errors in Raw State Preparation Circuits Recall Definition 6 for fault-tolerant encoding for CSS codes, which requires that any X- and Z-error occuring during encoding does not propagate to a same-type higher-weight error, separately. In general, we defined the weight of a discrete Pauli error as the minimum weight of any Pauli operator equivalent to it modulo multiplication with a stabiliser group. For the encoding of a specific state, this group extends the usual stabiliser group $\mathcal{S}$ of a QEC code (generated by some $n-k$ stabilisers) with the k logical operators fully specifying the state [67]. For example, when encoding the logical $|0\rangle_L^{\otimes k}$ state, the k logical Z-operators stabilise the final state. Hence, if a 'logical Z-error' were to occur, it would merely fix the state; equivalent to a weight-zero error. Even though this shows that logical Z-errors are per se not problematic for the encoding of the particular state itself, we still need to protect against Z-errors, since high-weight Z-errors can become problematic when using the state preparation routine (without decoding) as a primitive for other routines, such as Steane-type error correction [27, 67]. Keeping this technical detail in mind, in the following, we refer to propagated errors on the data qubits that violate the FT criterion in Definition 6 as *harmful errors*[67]. For example, for a distance-5 code and

[66] The Steane-$[[7,1,3]]$, Shor-$[[9,1,3]]$ and Color-$[[17,1,5]]$ codes serve as comparisons.

[67] Note that we use this term differently from [26], where such errors coincide with uncorrectable errors; here, a harmful error could also be a weight-s' error that is caused by some s errors, with $s < s' \leq t = \lfloor (d-1)/2 \rfloor$. Then, the harmful error violates the FT criterion, despite being correctable [27, Section V.A.].

$t = \lfloor (d-1)/2 \rfloor = 2$, a harmful error could be a propagated error of weight ≥ 2 resulting from a single error in the circuit, or a propagated error of weight ≥ 3 resulting from two errors in the circuit.

When considering a raw encoding circuit obtained from the GA optimisation in the previous section, our fault tolerance criterion is generally violated[68]. In particular, errors can propagate freely through the CX gates according to Figure 3, potentially leading to harmful errors.

Constructing Verification Circuits In Section 2.1.11, we already encountered verification circuits as a potential remedy to this uncontrolled error propagation. In general, a verification circuit is a set of stabiliser measurements, selected from the large set of all Pauli operators stabilising the state to be encoded. Since we work with CSS codes, we may handle X- and Z-errors, and therefore also their verification, separately, by considering the sets of harmful X- and Z-errors propagated by a given number of errors in the circuit, which is useful in the construction of verification circuits. For a treatment of the inner workings of verification circuits, we refer to [27], providing us with an automated method of synthesising efficient verification gadgets.

In [27], stabilisers are chosen such that for any harmful error, at least one of the stabilisers anti-commutes with the error, making it detectable via stabiliser measurements (cf. Section 2.1.4). This implies that to detect a set of harmful X-errors, we utilise Z-type stabilisers, and vice versa. Herein, verification works by post-selecting runs based on measurement outcomes; if at least one of the outcomes is -1, the run is discarded. The error sets in question can be computed efficiently by propagating a certain number of errors within the procedure through the circuit, reducing the propagated errors based on their equivalence with stabilisers, and omitting low-weight errors, as discussed in [27, Section V.A]. The task of finding a minimally-sized set of stabilisers to be measured, given a certain set of harmful errors, such that at least one stabiliser anti-commutes with each of the errors, is NP-complete [27, Theorem V.I]. However, `MQT-QECC` offers a SAT-based gate-optimal approach, as well as a computationally efficient heuristic method based on set covers, to enable verification circuit synthesis for small problem instances, given a raw CSS code preparation circuit. For our experiments, we use the optimal synthesis method[69] across all code-architecture pairs. Note that testing different architectures for the same code in general also yields different non-FT state preparation circuits, due to the adaptation to the specific mapping; thus, also the verification circuits differ as a result of the changed sets of harmful errors propagated by the respective optimised circuit.

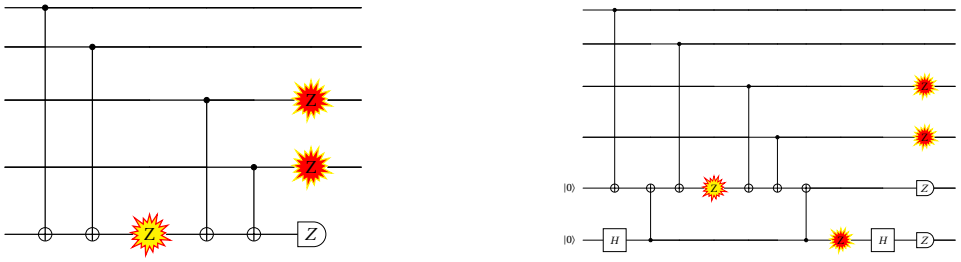
Adding a verification circuit does not come for free; since our fault tolerance criterion applies to the entire construction, i.e., the non-FT state preparation plus verification circuit, any combination of errors in state preparation and encoding circuit must not propagate to a harmful error on the data qubits *without being detected* by the verification circuit measurements (and thus being discarded)[70]. Now any verification circuit does not only detect harmful errors, but can also introduce such errors itself, since er-

[68] Our fault tolerance design principle allows for appending verification circuits in a DQC setting without violating fault tolerance, but does not make the raw circuits fault-tolerant itself.

[69] This synthesis method is gate-optimal in a *monolithic* setting.

[70] Generally, errors can propagate from ancilla and communication qubits onto data qubits; however, since any non-data qubits is effectively traced out of the systems via measurement, harmful errors themselves are only defined on data qubits (cf. Section 4.1.4).

rors can propagate via the indirect measurements. For example, consider the left panel of Figure 16, which depicts a potential weight-4 Z-type stabiliser measurement. Here, if a Z-error occurs after the second CX gate on the ancilla qubit, this single Z-error propagates through the remaining two CX gates onto the corresponding data qubits, and causes a weight-2 error modulo the stabiliser. Such harmful errors introduced by a verification circuit itself are known as *hook errors*. A solution to this adds so-called *flag qubits* (cf. [84]); these (flag) ancilla qubits are entangled with the stabiliser measurement ancillas to yield a (-1)-measurement outcome (i.e., 'flag' a harmful error) whenever an error on the stabiliser measurement ancilla propagates to a harmful error on the data qubits. As depicted in the right panel of the figure, the same Z-error on the first ancilla now also propagates to a Z-error on the flag ancilla, which is transformed to an X-error by the Hadamard and then detected (via Z-basis measurement). This allows us to also post-select on flag qubit measurements, in order to guarantee that the system did not suffer from harmful error introduced by the stabiliser measurements of the verification circuit themselves.



(a) Plain verification circuit with one ancilla for stabiliser measurement, in which a single Z-error (red outline) propagates to a harmful weight-2 error (yellow outline).

(b) Additional ancilla qubit to flag harmful errors; the Z-error (yellow) also propagates to an error on the flag qubit (red), yielding measurement outcome -1.

Figure 16: Weight-4 Z-type stabiliser measurement circuit, with and without flag qubit. Adapted from [157, Figure 3].

So far, we have described how a verification circuit (including flag qubits) is used to verify that a given number of errors in the entire circuit does not propagate to harmful errors without being detected. Due to our fault tolerance criterion (cf. Definition 6), we lastly need to make sure that this verification is achieved for harmful errors resulting from any s errors in the circuit, with $1 \leq s \leq t = \lfloor \frac{d-1}{2} \rfloor$. With the ingredients of stabiliser measurements and flag qubits, [27] devises a construction in which verification circuits are not only separated by X- and Z-type, but also concatenated with respect to the weight of the harmful error they verify. Specifically, verification circuits for catching harmful weight-1, weight-2, ..., weight-t are concatenated (for X- and Z-errors separately), thus consecutively verifying smaller subproblems[71]. When using the `MQT-QECC` tool, which implements the above components, in our implementation, we choose to always verify X-errors first (by measuring Z-type stabilisers), which allows us to include flag qubits for the X-type stabiliser measurements only.[72] [27]. Applying this

[71] This process is visualised in [27, Figure 1] and outlined in great detail in Section *V.A* of [27].

[72] In particular, we add potential Z-type hook errors to the fault set that is checked when verifying Z-errors by measuring X-type stabilisers.

sequential verification including flaq qubits to the optimised raw state preparation circuits from Section 4 then lets us detect any harmful X-or Z-type error in the encoding procedure.

Measuring the required stabilisers and flag qubits for X- and Z-errors of each weight, and post-selecting state preparation runs based on all measurement outcomes, we achieve fault tolerance of the remaining (accepted) runs. The fraction of accepted runs is an important metric regarding the efficiency of the protocol, and is commonly referred to as *acceptance rate*. Note that the acceptance of a state preparation run does not imply that the resulting state (prior to decoding) is error-free; in fact, the system could have suffered from a correctable low-weight error, a high-weight error or even a logical error. However, upon acceptance of a run, the fault-tolerant criterion from Definition 6 is satisfied, and no error on $s \leq t$ circuit locations propagated to a final error of weight larger than s. Figure 17 visualises these concepts.

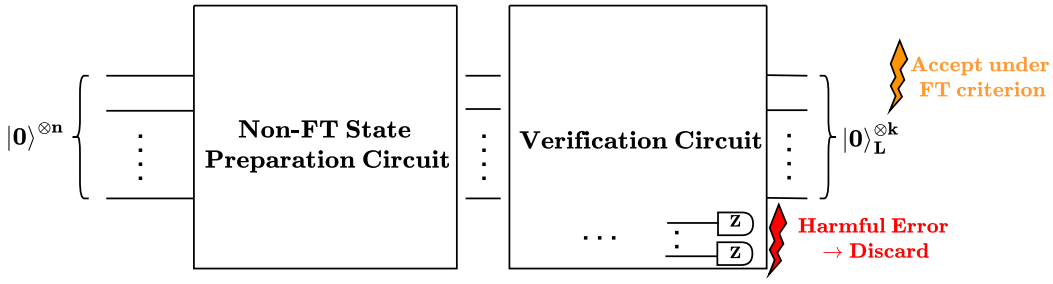


Figure 17: Fault-tolerant encoding procedure based on post-selection of runs, in which a raw non-FT state preparation circuit is concatenated with a verification circuit. For the entire mapping from physical to logical all-zero state, the fault tolerance criterion from Definition 6 is satisfied for all runs that are accepted. Adapted from [27, Figure 3].

For a concrete example of a verification circuit, Figure 18 presents the state preparation and verification circuits for the Trivariate Bicycle code (for the optimised circuit on two cores) side by side. Here, in the verification circuit, two Z-type stabilisers are measured first, followed by a (flagged) X-type stabiliser measurement. This verification circuit guarantees that no low-weight errors propagate to a harmful errors without being flagged by a (-1)-measurement in either of the ancillas or the flag qubit. Note that for this distance-3 code, the fault tolerance requirement simplifies substantially, since we only need to guarantee that any single error does not propagate to a harmful error of weight ≥ 2. Thus, we can consider state preparation and verification circuit separately: If one error in the state preparation circuit propagates to a harmful error, the (then noiseless) verification circuit catches it; if one error occurs in the verification circuit, it either propagates to a (harmless) weight-1 error on the data qubits (since hook errors are accounted for by the then noiseless flag qubits), or erroneously flips some verification measurement outcome. The latter case affects the efficiency of the protocol, since discarding a valid encoding run without harmful errors reduces the acceptance rate. Nonetheless, either of the two cases satisfies fault tolerance.

Adapting Verification Circuits For Type-II Systems In the above, we assumed that all CX gates in the state preparation circuit are performed in a monolithic setting. Since we are ultimately interested in distributed systems, we need to verify that the fault tolerance of the procedure in Figure 17 also translates to the Type-II DQC context, which

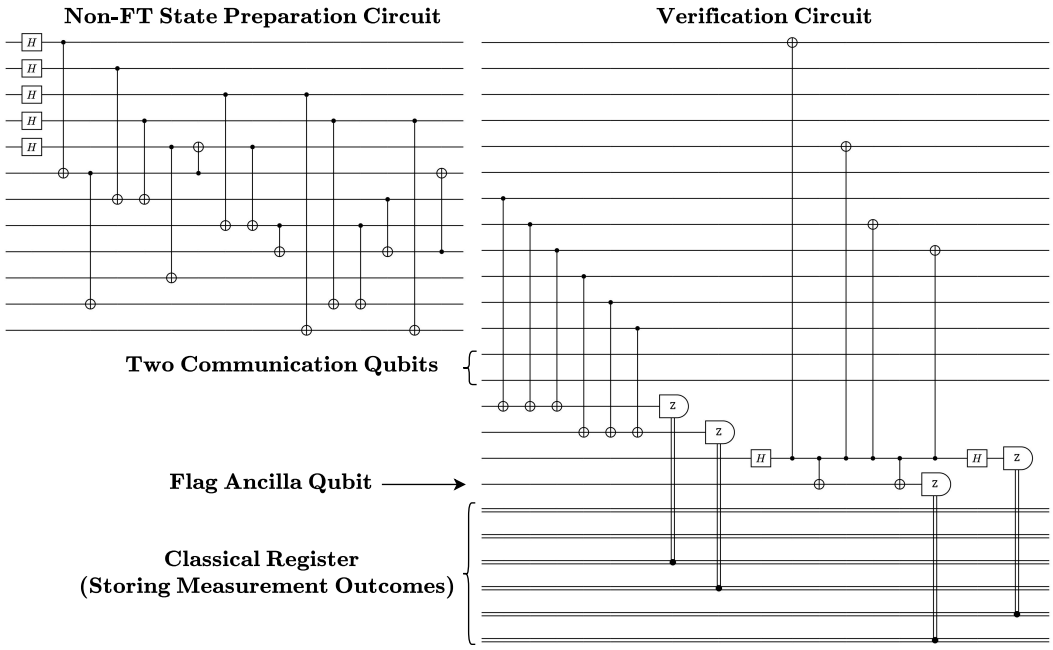


Figure 18: Example of full fault-tolerant state preparation routine, following the schematic in Figure 17, for the Trivariate Bicycle $[[12,2,3]]$ code on two cores. Non-FT state preparation circuit optimised for mapping on two cores, as displayed in Figure 12.

we are optimising for in Section 4. The key difference in a distributed setting is that we need to mediate certain gates via remote entanglement. For this, in Figure 9, we already saw how X- and Z-errors propagate through the telegate protocol of a remote $CX_{q_1\to q_2}$ gate[73]. In particular, we observed in Section 4.1.4 that any X-error (Z-error) on the data qubits q_1 or q_2 propagates according to Figure 3 through the control (target), and is therefore accounted for by the monolithic fault-tolerant construction. Furthermore, any X-error (Z-error) on either of the communication qubits propagates to precisely one X-error (Z-error) on the target q_2 (control q_1). Thus, for any telegate either in the non-FT state preparation circuit or in the verification circuit, errors on the communication qubits do not lead to an uncontrolled spread of errors, and can be represented as local (unpropagated) Pauli errors at the associated data-qubit locations; their subsequent propagation is already accounted for by the layered verification construction. Hence, the monolithic fault-tolerance construction from [27] is not violated by including telegates, and thus extends to both the distributed (non-FT) state preparation and verification circuits. Consequently, we can directly apply it to our Type-II DQC simulations.

Note that the argument above explicitly relies on the assumption that 1 telegate = 1 e-bit holds, which we promoted as design choice for optimising the non-FT state preparation circuit in Section 4. Thus, the very choice of our optimisation objective enables us to achieve fault tolerance in a Type-II setting by simulating the (originally monolithic) verification procedure [27] in a distributed setting.

Throughout our discussion of fault tolerance, based on Definition 6, we account

[73] In addition to the ancilla qubits of the verification circuit itself, also the communication qubits are valid error locations in the DQC setting.

for a noise model of X- and Z-errors. In particular, we consider error locations of such errors, and how they propagate in a circuit. While this treatment captures the property of fault tolerance to control the propagation of errors, it implicitly carries a claim about error probabilities: Assuming a simplistic error model of independent and uniform X- and Z-errors with error probability p on all data locations, Definition 6 states that for $s \leq t$, an error of probability on the order of p^s, i.e., an error on s error locations with probability p each, propagates to an error of weight at most s [80]. Yet, at the expense of deviating from this simplistic error model perfectly matching our definition of fault tolerance, we seek to assess the encoding procedure we developed in a realistic distributed setting. The design principle of fault tolerance then still stands for the simplified noise model, and we can evaluate to what extent it serves as a valuable proxy to the more realistic noise model. In this spirit, we next introduce the noise model we employ for our simulations.

5.2 Simulation Methods

The simulation of fault-tolerant state preparation encompasses many design choices, including the metrics of interest and the precise simulation and evaluation techniques. As a first piece in the set of simulation methods put to practice in this study, we discuss the particular error model we choose in this work to mimic a realistic DQC environment.

5.2.1 DQC Error Model

Quantum information stored and processed on real hardware suffers from many types of noise. The authors of [158] list three main sources thereof: (i) hardware infidelities, (ii) state preparation and measurement errors, and (iii) decoherence. Hardware engineering tackles these sources by providing higher-precision lasers or better cryogenics to improve gate or readout fidelities and limit interactions of a quantum system with its environment. Nonetheless, without quantum error correction, quantum hardware noise is too large for computation to be useful.

To model noise, we use a circuit-level noise model, based on a depolarising channel [27, 125, 44]; a depolarising channel is a quantum channel ϵ, parameterised by some noise parameter λ, as introduced below.

Definition 8: Depolarising Channel

A quantum depolarising channel is the parameterised map ϵ_λ^n on the space of n-qubit density operators, defined by

$$\epsilon_\lambda^n(\rho) = (1-\lambda)\rho + \lambda\frac{\mathbb{1}_d}{d},$$

where $d = 2^n$ is the dimension of the Hilbert space, and λ satisfies

$$0 \leq \lambda \leq 1 + \frac{1}{d^2-1}.$$

As such, this noise channel maps some quantum state, expressed via its density operator ρ, to a superposition of itself and the maximally mixed state $\mathbb{1}_d/d$. In this

work, we focus on the one-qubit ($d = 2$) and two-qubit ($d = 4$) case. To convert the abstract Definition 8 into a tangible model of error (probabilities), we perform some algebra.

For $d = 2$ (one qubit), we have

$$\epsilon^1_\lambda(\rho) = (1-\lambda)\rho + \lambda\frac{\mathbb{1}_2}{2} = (1-\frac{3\lambda}{4})\rho + \frac{\lambda}{4}(X\rho X + Y\rho Y + Z\rho Z), \tag{50}$$

where we have used the equality $\rho + X\rho X + Y\rho Y + Z\rho Z = 2\mathrm{Tr}(\rho)\mathbb{1}_2 = 2\mathbb{1}_2$. Similarly, for $d = 4$ (two qubits), we obtain

$$\epsilon^2_\lambda(\rho) = (1-\lambda)\rho + \lambda\frac{\mathbb{1}_4}{4} = (1-\frac{15\lambda}{16})\rho + \frac{\lambda}{16}\sum_{P_1,P_2\in\mathcal{P}_2\setminus\{P_1=P_2=\mathbb{1}_2\}}(P_1\otimes P_2)\rho(P_1\otimes P_2), \tag{51}$$

where we have used the equality

$$\sum_{P_1,P_2\in\mathcal{P}_2}(P_1\otimes P_2)\rho(P_1\otimes P_2) = 4\mathbb{1}_4.$$

Now define the error probabilities $p_{1q} = \frac{3\lambda}{4}$ and $p_{2q} = \frac{15\lambda}{16}$ (in general, we have $p_{nq} = \frac{(d^2-1)\lambda}{d^2}$) for the respective λ parameter in Equations 50 and 51, which we can consequently rewrite as

$$\epsilon^1_\lambda(\rho) = (1-p_{1q})\rho + \frac{p_{1q}}{3}(X\rho X + Y\rho Y + Z\rho Z), \tag{52}$$

and

$$\epsilon^2_\lambda(\rho) = (1-p_{2q})\rho + \frac{p_{2q}}{15}\sum_{P_1,P_2\in\mathcal{P}_2\setminus\{P_1=P_2=\mathbb{1}_2\}}(P_1\otimes P_2)\rho(P_1\otimes P_2). \tag{53}$$

Note that we now have $0 \le p_{1q}, p_{2q} \le 1$, where the above equations indeed let us interpret p_{1q} and p_{2q} as error probabilities for the single-qubit or two-qubit error channel; with probability $p_{1q}/3$, any of the three single-qubit Pauli errors in $\{X, Y, Z\}$ occurs, and with probability $p_{2q}/15$, any of the 15 Pauli errors among the two-qubit operators $\{XI, XX, XY, \ldots, ZY, ZZ\}$ occurs. In order to emulate this noise channel in our simulations without having to abandon the Clifford formalism for a density operator simulation, we sample these Pauli errors for each error location, according to the error probabilities parameterising the single- and two-qubit depolarising channel. This lets us mimic all three types of noise introduced earlier, as summarised in below.

Procedure 1: Modeling Noise with Depolarising Channels

- To simulate single- and two-qubit gate infidelities, we apply the respective depolarising channel (cf. Equations 52 and 53) after a perfect gate by sampling a single- or two-qubit Pauli error, respectively.
- State preparation and measurement (SPAM) errors are commonly represented via bit-flip errors [27, 125]. In our simulations, we initialise qubits in the physical $|0\rangle$ state, and measurements are performed in the computational basis. When applying a depolarising channel (cf. Equation 52) with noise probability p_{1q} (after qubit initialisation or before measurement), we obtain a bit-flip error with the effective probability $2p_{1q}/3$, since

only X- and Y-errors (but not Z-errors) map between between $|0\rangle$ and $|1\rangle$. In order to apply a state preparation or measurement error with *effective* error probability p_{init} or p_{meas}, we thus apply a depolarising channel with noise probability $p_{1q} = 3p_{\text{init}}/2$ or $p_{1q} = 3p_{\text{meas}}/2$, respectively [a].

- As another instance of state initialisation, we model the initialisation of Bell pairs as part of the telegate protocol with a (two-qubit) depolarising channel (cf. Equation 53), i.e., we post-apply one of the 15 two-qubit Pauli errors with probability p_{2q}. Note that the first Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}\Big(|00\rangle + |11\rangle\Big)$ is stabilised by the (non-trivial) Pauli operators XX, YY and ZZ (up to global phase), which hence do not affect the initialisation. We thus apply a depolarising channel with noise probability $p_{2q} = 15p_{\text{Bell}}/12$, such that each of the (twelve) non-stabilising errors occurs with probability $p_{\text{Bell}}/12$, achieving an *effective* Bell state initialisation error rate of $12 \cdot p_{\text{Bell}}/12 = p_{\text{Bell}}$.
- We model decoherence by applying a symmetric depolarising channel (cf. Equation 52) of a given noise strength to idling qubits, which will further be elaborated on in Section 5.2.2.

[a] Note that we need to ensure that $3p/2$ is still a valid error probability, i.e., $3p/2 \leq 1$. For the small error rates in our simulation, this is satisfied.

In order to motivate our simulation with real hardware data, we consider the ion trap hardware specifications of the `Quantinuum H1-1` computer, released in 2022. This quantum computer hosts twenty fully connected qubits on a single chip, and is thus reasonably close to the system sizes (per core) we consider in this work[74]. The specifications listed in Table 2 are taken from the respective system performance validation documentation [160].

Error type	Infidelity
Single-Qubit Gate Error	$1.8 \cdot 10^{-5}$
Two-Qubit Gate Error	$9.7 \cdot 10^{-4}$
$\|0\rangle$ State Preparation and Measurement Error	$1.2 \cdot 10^{-3}$
Memory Error per Depth-1 Circuit Time	$2.2 \cdot 10^{-4}$

Table 2: Infidelities per error type for the `Quantinuum H1-1` quantum computer, as reported in [160].

Next, we collect experimental state-of-the-art results for remote-entanglement fidelities, as they concern the telegates in our simulations. In this context, [161] reports a Bell pair creation fidelity of 94% at an average rate of $182s^{-1}$, dating back to 2020. In a more recent experiment, a remote CZ gate is performed with a fidelity of 86% [100], drawing upon the generation of a remotely entangled Bell pair $|\psi^+\rangle = \frac{1}{\sqrt{2}}\Big(|01\rangle + |10\rangle\Big)$ between networking (communication) qubits of fidelity close to 97%. The correspond-

[74] Whereas the `H1` system matches our system sizes, more recent machines (such as the 98-qubit `Quantinuum Helios-1` [5]) exhibit comparable infidelities [159]. Expecting that early-stage QEC experiments on Type-II architectures are performed on small systems, the choice of noise model based on the `Quantinuum H1-1` computer in Table 2 is sensible.

ing Bell state generation rate for this setup is listed as $9.7s^{-1}$. In [162], an entanglement fidelity of $\approx 97\%$ at a rate of $0.35s^{-1}$ is reported, while [163] achieves a fidelity of $\approx 94\%$ at a high rate of $250s^{-1}$. The exact fidelities and rates depend on the choice of qubit encoding (e.g., polariston vs. time-binning) and the ion or atom cooling mechanism, among others, but the authors of [162] "show that fundamental limits due to atomic recoil still allow fidelities in excess of 99.9%".

Since the infidelities reported above are proportional to error probabilities, we find that the qualitative distinction between current intra-core and inter-core operations also materialises as a quantitative difference. For our DQC simulation, we thus define distinct error probabilities p and p_{Bell} for intra- and inter-core operations, respectively. Here, p_{Bell} refers to the Bell state initialisation error rate, which is simulated via the two-qubit depolarising channel, as introduced in Procedure 1. The intra-core operations, on the other hand, are further quantified based on Table 2, from which we estimate ratios between error probabilities for different error types. In particular, we see that two-qubit gate errors and SPAM error are on the same order of magnitude, which we denote with $p = p_{\text{two-qubit gate}} = p_{\text{init}} = p_{\text{meas}}$[75]. The single-qubit gate error probability is separated from these error types by almost two orders of magnitude, which we reflect by choosing $p_{\text{single-qubit gate}} = p/100$. And lastly, we interpret the memory error per depth-1 circuit time from Table 2 as our idling error probability, which is approximately one order of magnitude noisier than the reference operations, yielding $p_{\text{idle}} = p/10$. Table 3 summarises the error probabilities of our DQC noise model[76], which we utilise as inputs to the one- and two-qubit depolarising channel, based on the discussion in Procedure 1.

Error type	Error probability
Single-Qubit Gate Error $p_{\text{single-qubit gate}}$	$p/100$
Two-Qubit Gate Error $p_{\text{two-qubit gate}}$	p
Physical $\|0\rangle$ Initialisation Error p_{init}	p
Z-basis Measurement Error p_{meas}	p
Idling Error p_{idle}	$p/10$
Bell State Initialisation Error	p_{Bell}

Table 3: Error probabilities per error type for implementation of circuit-level noise model via depolarising channels.

Given this error model, including a relative weighting of error types and two variable noise parameters, we sweep p and p_{Bell} over multiple values[77] in our DQC simulations, in order to assess how realistic noise impacts the state preparation routine.

5.2.2 Scheduling Circuit Operations

As part of our circuit-level noise model, we take into account that idle qubits experience decoherence. Therefore, we need to develop a notion of what *idling* means. For

[75] Here, initialisation and measurement error are treated separately, details of which can be found in Appendix B.16.

[76] We omit the effect of crosstalk in this discussion, and we assume that the noise specifications are identical for data and communication qubits, with two-qubit gate error probabilities not being impacted by the type of ion species at control and target.

[77] For this, we restrict the range of p_{Bell} to a higher-error regime than the range of p, based on the significant difference in noise levels on current quantum systems.

this, we build up layers of quantum circuit operations via *As-Soon-As-Possible* (ASAP) scheduling; gates in the same ASAP layer can be executed in parallel during one timestep, as depicted in Figure 19. Then the idle qubits within an ASAP layer are precisely those data or ancilla qubits that do not participate in any quantum operation (intra- or inter-core gates or measurement) in this layer.

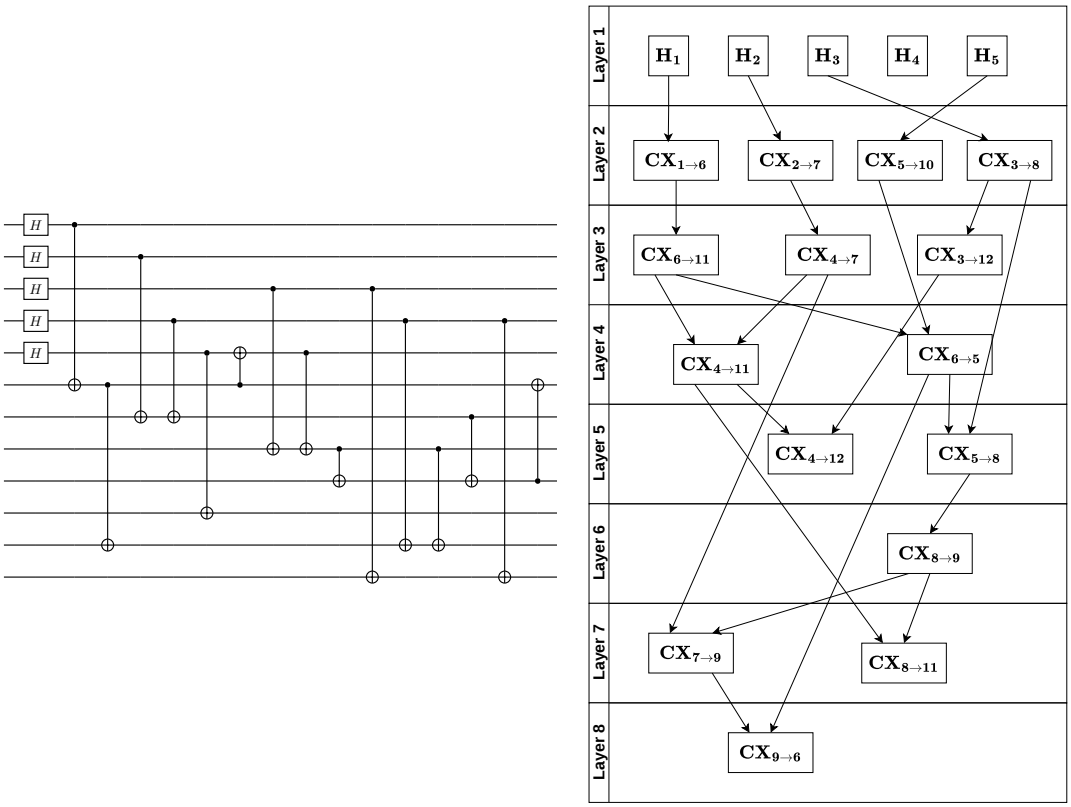
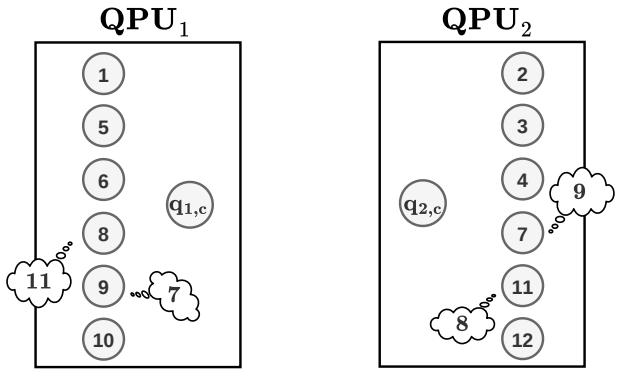


Figure 19: Example of As-Soon-As-Possible (ASAP) scheduling of quantum circuit. Left: Quantum circuit to be executed (here: non-FT state preparation circuit for Trivariate Bicycle-[[12,2,3]] code on two cores). Right: ASAP layers, where gates are scheduled according to earliest availability of all qubits affected by the gate. Gates within each layer can be executed in parallel.

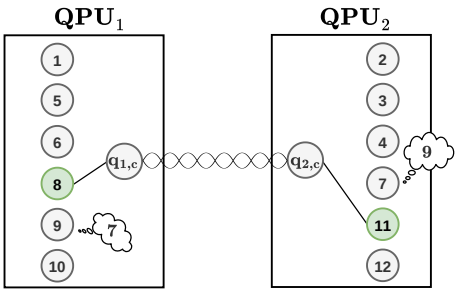
Given this scheduling mechanism, we now traverse each ASAP layer to execute the corresponding operations, and track the respective idle qubits. Since we are working with a quantum network, however, there is an additional constraint: The communication qubits used to mediate remote gates need to be available. A straightforward way to achieve this is to add as many communication qubits as data qubits to each core. This mitigates any conflict of shared communication qubit utilisation, but creates a large overhead in qubit count per core. On the other hand, reducing this count as much as possible by only allowing one communication qubit per core increases the number of scheduling conflicts. We opt for an intermediate approach, placing just as many communication qubits in each core as there are other cores. This is in line with our assumption of a limited number of communication qubits per core in Type-II systems (cf. Concept 2), yet guarantees that for each layer, any two cores can perform exactly one telegate in a given timestep. For simplicity, we assume that only the dedicated

communication qubit for the respective target core may be used to perform a telegate. Now if within an ASAP layer, we find multiple remote gates between some pair of cores, these remote gates need to be executed in consecutive rounds, as depicted in Figure 20 for the example of the seventh layer in Figure 19.

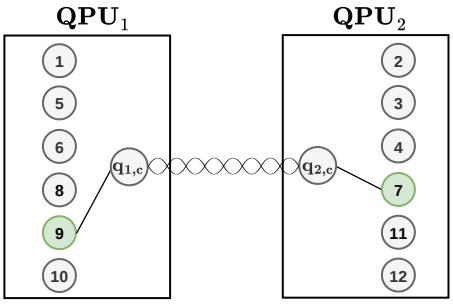
Greedily scheduling telegates based on the availability of communication qubits, we denote the number of consecutive Bell state creation wait-time rounds within a scheduled layer with n_{tele}. For example, in Figure 20, we have $n_{\text{tele}} = 2$ for Layer 7, since both telegates require access to the same communication qubit pair. It thus becomes clear that idle qubits should experience different levels of depolarising noise in our simulation, depending on the number of n_{tele} telegate rounds in an ASAP layer.



(a) Scheduled Telegates for ASAP layer



(b) Telegate Round 1



(c) Telegate Round 2

Figure 20: Telegate rounds for ASAP layer 7 from Figure 19, based on qubit mapping in Figure 8. Both remote gates $CX_{8\to11}$ and $CX_{7\to9}$ require access to the pair of communication qubits $(q_{1,c}, q_{2,c})$. Inspired by [103, Figure 7].

To gauge just how different these idling error probabilities are, we again refer to real hardware values. We already encountered that the average Bell state generation time for photonically-connected ion traps, ranging from 4 ms in [163] to 2.857 s in [162], imposes a large overhead on our telegate operations. At the same, ion trap systems generally also exhibit considerable gate times already for intra-core gates[78]. In [5], the authors introduce the concept of a *depth-1 time* — the characteristic time on which

[78]The large gate time, in contrast to the high fidelity of gate operations, is one of the main limiting factors for ion trap systems

the memory error in Table 2 is reported — as the time it takes to randomly pair and perform single- and two-qubit gates on all qubits in a single layer, given some ion trap. For the `Quantinuum Helios` model, this time is reported as 55 ms per layer [5]. For the smaller `IonQ Aria` ion trap system, a raw two-qubit gate speed of 600 µs is reported [164]. Based on this data, we roughly estimate that on near-term machines, the Bell state creation time exceeds the two-qubit gate time by one order of magnitude.

We then model the accumulated idling error probability for a complete telegate round (including Bell state generation and local operations of the telegate protocol in Figure 5), denoted as $p_{\text{idle}_{\text{tele}}}$, according to this timing mismatch by applying $\eta = 12$ instances of single-qubit noise[79] with the standard idling error probability $p_{\text{idle}} = p/10$. To gain some efficiency in our simulations, we approximate $p_{\text{idle}_{\text{tele}}}$ to first order in p as

$$p_{\text{idle}_{\text{tele}}} \approx 1 - (1 - p_{\text{idle}})^{\eta}, \tag{54}$$

and sample idling errors for one telegate round from the single-qubit depolarising channel with error probability $1 - (1 - p_{\text{idle}})^{\eta}$ (cf. Appendix B.17). This yields the following case distinction for idling depolarising noise.

Procedure 2: Idling errors

For each ASAP layer, we determine the number of n_{tele} telegate rounds, and apply idling noise based on the following cases:

- If $n_{\text{tele}} = 0$, there are no telegates in the scheduled layer, and we set the idling error probability to $p_{\text{idle}} = p/10$;
- Otherwise, if $n_{\text{tele}} > 0$ telegate rounds occur within a layer, we sample noise from a depolarising channel with error probability $1 - (1 - p_{\text{idle}})^{\eta}$ for n_{tele} consecutive times.

In addition, we apply idling errors to the data qubits involved in a telegate in a layer[a], both for the wait time for their own communication qubit entanglement, as well as for wait times accrued from other telegates in the same layer due to scheduling conflicts[b].

[a] Only communication qubits are exempt from idling noise, since we assume that any idling noise during Bell state generation is absorbed in the elevated error rates of p_{Bell}.

[b] Details of this are provided in `https://github.com/timneumann1/DQCode/blob/main/src/dqc_simulator.jl`.

Overall, we find that even for a homogeneous quantum network, gate scheduling based on limited availability of communication qubits leads to a significant accumulation of idling noise in the system, especially in the presence of (non-parallelisable) telegates. Our circuit-level noise model herein does not only incorporate varying error probabilities per error type, but also accounts for realistic scheduling delays.

[79] Our choice of $\eta = 12$ time steps as duration of a telegate captures our estimate of an order of magnitude time separation between Bell state generation and two-qubit gate time. Similarly, [125] assumes that the "number of attempts required to initialise a Bell state is 5 gate cycles". Here, our estimate includes the additional time of local operations (single- and two-qubit gates, measurements, classical communication and conditional operations) in the telegate protocol.

5.2.3 Lookup Table Decoding

In order to evaluate the quality of our state preparation routine, we are mainly interested in the logical initialisation error rate of the logical $|0\rangle_L$ state per logical qubit. Recall that our fault-tolerant state preparation asserts that up to $t = \lfloor (d-1)/2 \rfloor$, any $s \leq t$ errors in the circuit do not propagate to harmful errors of weight larger than s. Thus, the errors resulting from up to t error locations in the encoding routine will always be correctable after post-selection of runs. Since the logical error rate explicitly depends on the correction of our state based on an extracted syndrome, we perform one round of perfect, i.e., noiseless, syndrome extraction and decoding after completing the verification circuit (at the end of the procedure in Figure 17, as captured by Figure 22). This is acceptable since in a real experiment, our state preparation does *not* require a syndrome extraction (which would necessarily be noisy on a real system) directly after encoding, as explained in Section 3.

Syndrome extraction provides us with the information needed to assess the fault tolerance of the routine, since it enables a decoder to infer the most probable error that occurred on the data. In this work, we utilise a so-called *Lookup Table Decoder*, similar to [49]. For each CSS code in question, we define two lookup tables, one for X- and Z-errors; a lookup table is a dictionary mapping all possible errors up to weight t at the end of the encoding circuit to the syndrome they produce when measuring the respective stabilisers (Z-type stabiliser measurement to detect X-errors and vice versa). For example, consider the Z-type table decoder of the Trivariate Bicycle-$[[12,2,3]]$ code on twelve qubits. This table decoder stores all possible weight-1 X-errors and the corresponding six-bit syndrome they produce when measuring the six Z-type stabilisers. For one of the twelve single-qubit error locations, it contains the mapping

$$[0,0,1,0,0,0,0,0,0,0,0,0] \mapsto [1,0,1,0,0,0],$$

indicating that a X_3 error at the end of the circuit triggers the first and third Z-type parity check. The X-type table decoder works analogously, indicating which Z-type error is present at the end of the circuit.

The guarantee of a lookup table decoder is that up to weight-t errors, it will reliably map the measured syndrome to the $\leq t$ error locations at the end of the circuit, allowing us to correct the error. The drawback of such a decoder is that populating it with all maps of errors to syndromes is computationally intractable; for n qubits and distance d, there are

$$\sum_{i=1}^{\lfloor (d-1)/2 \rfloor} \binom{n}{i} \tag{55}$$

error locations to be considered for both table decoders, separately. But since we are working with distance-3 and distance-5 codes of moderate qubit size in this study, a CSS Lookup Table is a viable choice of decoder.

5.2.4 Metrics

With access to fault-tolerant state preparation circuits, an error model as well as scheduling and decoding routines, we are now in the position to simulate and assess the encoding procedure in a DQC setting. In this section, we describe the logical initialisation error rate and acceptance rate as primary metrics of interest, and also introduce

the pre-decoding error rate as well as a framework for resource comparison. In Section 5.3, we then present the results of extracting these metrics from the simulations of differnet codes and architectures.

Logical Initialisation Error & Acceptance Rate. Since we are interested in the fault-tolerant encoding of a logical state in a noisy DQC setting, we want to make claims about the error rate associated with this logical encoding procedure. Whereas [27] and [26] refer to this error probability simply as Logical Error Rate, we define the *Logical Initialisation Error Rate*, in order to demarcate it from error rates usually encountered in memory experiments[80] [125].

Definition 9: Logical Initialisation Error Rate

The *Logical Initialisation Error Rate* (LIER) of a logical qubit is the the probability of the logical qubit not being in the desired state after both the corresponding state preparation routine and a round of perfect syndrome measurements and error correction, conditioned on the run being accepted in the verification post-selection process.

The focus of this work lies on preparing the logical $|0\rangle_L^{\otimes k}$ state. In this light, we can define the LIER_i as the true probability of the i^{th} logical qubit to be in the state $|1\rangle_{L,i}$ after encoding, post-processing, noiseless syndrome extraction and correction[81]. How can we extract this probability from our simulation? As a first step, we introduce the Bernoulli random variable

$$X_i = \begin{cases} 1, & \text{if } |\psi_c\rangle_{L,i} = |1\rangle_{L,i} \\ 0, & \text{if } |\psi_c\rangle_{L,i} = |0\rangle_{L,i}\,, \end{cases} \tag{56}$$

where $|\psi_c\rangle_{L,i}$ is a corrected state after perfect syndrome extraction and correction, conditioned on being accepted by the verification circuit. Note that this matches our simulation: Firstly, $|\psi_c\rangle_{L,i}$ exhibits randomness due to the stochasticity of the noise process underlying the state preparation; secondly, the combination of our circuit-level depolarising channel error model and decoder corrections can only ever induce logical Pauli errors on the ideal $|0\rangle_L^{\otimes k}$ state, so any corrected state $|\psi_c\rangle$ is (up to global phase) a product state of either the logical $|0\rangle_{L,i}$ or the logical $|1\rangle_{L,i}$ state in each logical qubit. This follows since each of the sampled errors from the circuit-level noise model is some Pauli gate, and thus any final error on the logical zero state is a member of the Pauli group; upon correction with a Pauli gate, such error becomes either an element of the stabiliser group, or a logical Pauli operator. Then note that LIER_i is precisely the expectation value of X_i over multiple (post-selected) quantum trajectories, i.e.,

$$\text{LIER}_i = \mathbb{E}[X_i] = \mathbb{P}\Big(|\psi_c\rangle_{L,i} = |1\rangle_{L,i}\Big). \tag{57}$$

From this perspective, it becomes clear that the logical initialisation error rate for the $|0\rangle_L^{\otimes k}$ state measures the rate at which logical X- or Y errors occur on the i^{th} logical

[80] In memory experiments, logical error rates are related to the probability of maintaining a logical state, not mapping into it.

[81] I.e., we measure errors in the i^{th} logical Z-observable.

qubit in the encoding procedure[82]. We can now approximate this logical initialisation error rate with the sample mean

$$\overline{\mathrm{LIER}_i} = \frac{1}{n_{\mathrm{acc}}} \sum_{s=1}^{n_{\mathrm{acc}}} X_i^{(s)}, \tag{58}$$

where we assume an i.i.d. sequence of n_{acc} random variables $X_i^{(s)}$. Equation 58 can be manipulated to read

$$\overline{\mathrm{LIER}_i} = \frac{1}{n_{\mathrm{acc}}} \sum_{s=1}^{n_{\mathrm{acc}}} \mathbb{P}\left(|\psi_c\rangle_{L,i}^{(s)} = |1\rangle_{L,i}\right) = \frac{1}{n_{\mathrm{acc}}} \sum_{s=1}^{n_{\mathrm{acc}}} \left\langle \psi_c^{(s)} \middle| \frac{1 - Z_{L,i}}{2} \middle| \psi_c^{(s)} \right\rangle, \tag{59}$$

where the probability in the second term is either 0 or 1, and $|\psi_c\rangle_{L,i} \in \left\{|0\rangle_{L,i}, |1\rangle_{L,i}\right\}$[83]. Equation 59 now gives us real traction on estimating the true LIER_i, since we sum over terms that are (rescaled) expectation values $\langle\psi_c^s|Z_{L,i}|\psi_c^s\rangle$ of logical Z-operators[84]. In practice, measuring the respective logical operators of the corrected states and counting the -1 outcomes of the ancilla measurements yields the desired estimator $\overline{\mathrm{LIER}_i}$. A more detailed treatment of Equation 59 and its implications can be found in Appendix B.18. Note how this relates to our interpretation of the LIER_i via logical errors: We measure -1 in the logical $Z_{L,i}$-observable precisely if a logical X_i- or Y_i-error is present at the end of the circuit, since these logical errors anticommute with the logical observable. Our overall strategy for obtaining the $\overline{\mathrm{LIER}_i}$ is summarised in Procedure 3.

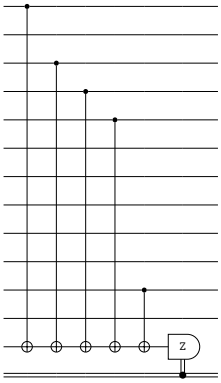


Figure 21: Example of indirect measurement of logical $Z_{L,1} = Z_1Z_3Z_4Z_5Z_{11}$ operator for Trivariate Bicycle-[[12,2,3]] code.

Procedure 3: Extracting $\overline{\mathrm{LIER}_i}$ from DQC simulations

In order to obtain $\overline{\mathrm{LIER}_i}$, we proceed by simulating a number of n_{traj} noisy DQC trajectories, where (Pauli) noise is sampled for each error location. We then discard or accept the final state depending on the measurements of the verification

[82] Technically, every logical Y-error is equivalent to a logical X-error since (up to global phase) both errors differ only by a logical Z-error, which stabilises the ideal target state.

[83] We dropped the $_L$ index in the expectation value of Equation 59 to avoid visual clutter, but the corrected state $|\psi_c^{(s)}\rangle$ is guaranteed to be a logical state.

[84] The equation features the expectation value of the projection operator $\frac{1-Z_{L,i}}{2}$, which we obtain by measuring $\langle\psi_c^s|Z_{L,i}|\psi_c^s\rangle$ via indirect stabiliser measurements such as depicted in Figure 21, and then scaling it accordingly.

circuit, and denote the overall estimator of the acceptance rate

$$\overline{\mathrm{AR}} = \frac{n_{\mathrm{acc}}}{n_{\mathrm{traj}}}, \tag{60}$$

where n_{acc} denotes the number of accepted runs. Next, for each of the accepted states, we perform noiseless syndrome extraction and correct the state based on the Lookup Table. For decoding, there are three different cases (cf. [25]):

1. The decoder cannot come up with an error guess $\hat{E}$ for the measured syndrome, such that no correction is applied.
2. The decoder finds a correction $\hat{E}$ that correctly undoes the error, up to stabiliser equivalence. Here, correctly uncomputing an error E via $\hat{E}E = \mathbb{1}$ also correctly uncomputes some other error $E' = ES$ for $S \in \mathcal{S}$ with the same syndrome, due to error degeneracy[a].
3. The decoder identifies a correction, but performing it completes a logical operation, i.e., $\hat{E}E \in \mathcal{C}(\mathcal{S}) \setminus \mathcal{S}$.

After decoding, we (noiselessly) measure the logical Z-operators for all k logical qubits, sum all the accepted runs for which either (i) the measurement outcome is -1 for the i^{th} such logical operator measurement, or for which (ii) the decoder does not infer an error guess (in which case we cannot map back to the codespace), and divide by the number of accepted runs.

[a] We then have $\hat{E}E' = \hat{E}ES = S$, where $\mathcal{S}$ fixes the state and is, by definition, equivalent to the identity up to multiplication with stabilisers.

While Procedure 3 captures the conceptual idea of obtaining our estimators, in our `DQCode` implementation we post-process error guesses and logical measurement outcomes to determine errors on the respective i^{th} logical Z-observable without actually applying a correction (similar to Pauli frame tracking). A detailed documentation of this alternative but equivalent procedure can be found in the codebase[85].

The above formalism also provides us with an estimate of the standard deviation of our sample mean $\overline{\mathrm{LIER}_i}$ (cf. Appendix B.19), which is captured by

$$\sigma\left[\overline{\mathrm{LIER}_i}\right] = \sqrt{\frac{\overline{\mathrm{LIER}_i}(1-\overline{\mathrm{LIER}_i})}{n_{\mathrm{acc}}}}.$$

Since the error in our estimate scales with $1/\sqrt{n_{\mathrm{acc}}}$, the number of overall samples n_{traj} implicitly determines the quality of our estimator $\overline{\mathrm{LIER}_i}$; however, we cannot control how many runs are actually post-selected. While this motivates to set n_{traj} as large as possible, we also need to take into account that we are performing noise sweeps over $24 \times 24 = 576$ pairs (p, p_{Bell}), imposing a significant multiplicative factor for computational runtime. Thus, for all our simulations, we choose to set $n_{\mathrm{traj}} = 500{,}000$, which

[85] Please refer to https://github.com/timneumann1/DQCode/blob/main/src/dqc_simulator.jl#L413-462 for an explanation. The implementation therein draws heavily upon the decoding pipeline outlined in https://github.com/QuantumSavory/QuantumClifford.jl/blob/2664eba07a461441ea051a17cad7c725f9576176/src/ecc/decoder_pipeline.jl (accessed 27.06.2026), which uses the concept of a `faults_matrix` to track which physical X- and Z-errors cause logical errors on the respective operators.

lets us explore interesting noise regimes; for example, in Section 5.3 we will see that many codes exhibit pseudothresholds.

Definition 10: Pseudothreshold for LIER

The pseudothreshold lies at the intersection of the physical initialisation error rate p and the average logical initialisation error rate

$$\text{LIER} = \frac{1}{k}\sum_{i=1}^{k} \text{LIER}_i \tag{61}$$

across all logical qubits, i.e., at $p = p_{\text{th}}$ satisfying

$$p_{\text{th}} = \text{LIER} = \frac{1}{k}\sum_{i=1}^{k} \text{LIER}_i, \tag{62}$$

where each LIER_i depends on both p and p_{Bell}.

We would like to stress that in Definition 10, the physical initialisation error rate p is the effective error rate for state initialisation, as discussed in Procedure 1, allowing for a direct comparison. The estimator $\overline{\text{LIER}}$ of the overall logical initialisation rate, as introduced in Definition 10, is the sum of the individual estimators, with its variance being adapted accordingly (cf. Appendix B.19). We are particularly interested in the behaviour of the LIER in the regime of the pseudothreshold, while we also need to ensure that the logical error rate does not fall far below $2\cdot10^{-5}$, such that we can still expect a couple ($\geq 5\cdot10^5 \times 2\cdot10^{-5} = 10$) of errors to occur. In order to reduce runtime, we dynamically check the number of logical errors already accumulated during a run, and perform an early stop once the mean number of such errors per logical qubit has surpassed 500 errors. The $\overline{\text{LIER}}$ is then determined based on this truncated sequence of trajectories at the cost of a small loss in accuracy in the high-error-rate regime, which is acceptable.

Besides the logical intitialisation error rate, we also extract the fidelity F between the resulting circuit state and our target $|0\rangle_L^{\otimes k}$ state from the simulation. This is achieved via squaring the overlap between the states[86] for accepted runs. In our specific case, the fidelity measures whether *all* k logical Z-observables are error-free, where logical Z-observables again yield deterministic measurements of 1 or -1 after correction. Thus, we expect F to lower bound the logical initialisation success rate; $F \leq 1.0 - \text{LIER}$.

In Section 5.3, we will assess whether our DQC state preparation routine exhibits logical initialisation error rates that indicate fault tolerance, whether and at which noise levels we obtain pseudothresholds, and how large typical acceptance ratios are. For ease of notation, we will henceforth drop the estimator notation, and refer to our estimators $\overline{\text{LIER}_i}$, $\overline{\text{LIER}}$ and $\overline{\text{AR}}$ simply as LIER_i, LIER and AR.

Pre-Decoding Error Rate. At many points in this section, we have already encountered the special role of Z-errors in the state preparation routine. Such errors cannot induce an error on the logical Z-observable of the target $|0\rangle_L^{\otimes k}$ state, but we still seek

[86] The overlap, or inner product, of two stabiliser states can be determined based on [34], which is implemented in the `dot` function at https://github.com/QuantumSavory/QuantumClifford.jl/blob/66b32759c1105896b474f10fe6e201458771d901/src/linalg.jl#L66.

to limit their propagation to harmful Z-errors in view of making our encoding a fault-tolerant starting point for consecutive operations[87]. This raises the question of how to obtain a proxy of the logical Z-error rate, when such errors technically do not even exist. The `MQT-QECC` library exposes a method based on Steane-type EC that allows us to extract some information about Z-error propagation via an additional ancillary system. Also the authors of [67] make use of this technique when assessing their fully fault-tolerant scheme. We do not translate this approach to our distributed setting, but instead consider *pre-decoding* error rates, both of X- and Z-type. Even though raw error rates before correction generally tell little about the logical error rate (since pre-decoding errors do not distinguish between correctable and uncorrectable errors), the rationale behind this is to obtain a proxy for how similar the spread of (physical) X- and Z-errors through the circuit is.

For this, we consider the measurement outcomes of the X-type stabilisers for the pre-decoding Z-error rate, and of the Z-type stabilisers (including the logical operators) for the pre-decoding X-error rate, similar to [87]. This way, we leverage anti-commutativity to identify which kind of physical error impacts the ideal logical all-zero state before decoding. Note that the pre-decoding error rate is obtained conditioned on accepted runs. The results of this analysis are presented in Section 5.3.

Resource Estimate. Whereas this study focuses on unitary encoding circuits appended with verification gadgets, we would be remiss not to compare our work to distributed stabiliser measurement as alternative method of encoding a quantum state. We already discussed the overhead associated with making such measurements fault-tolerant, and to some extent, the verification circuits in this work resemble the method of measuring all stabilisers, discussed in Section 2.1.11. However, the main difference is that the number of measurements needed here is generally much smaller. Thus, we focus on a comparison of resource estimates between methods. Throughout, we make favourable assumptions towards the measurement-based approach. For example, the authors of [67] state that a fault-tolerant measurement-based scheme "repeats the measurement $\lfloor d/2+1 \rfloor$ times and restarts upon the detection of any error", we compare our entire encoding procedure with only one layer of (distributed) stabiliser measurements. The reason for this is that certain codes admit single-shot decoding (cf. Section 2.1), and under the prospect of single-shot state preparation, we aim to provide a conservative comparison. To encode the logical zero state, we furthermore only measure X-type stabilisers, and for the Multivariate Bicycle codes, we use the low-weight stabilisers obtained from the construction given in Section 2.1.8 (rather than the canonical stabilisers).

Overall, we compare gate counts (single-qubit gates, intra-core two-qubit gates and telegates), the number of measurements and ancilla qubits required for each state preparation method, and the depth of the circuit in terms of (telegate) rounds. When analysing the encoding procedures for depth, we apply the same scheduling methods to either method to ensure commensurability; in particular, we firstly determine ASAP layers of operations based on the data qubits, and secondly the required telegate round overheads based on communication qubit conflicts, as outlined in Section 5.2.2. We then sum the total number of CX and telegate rounds, respectively, across layers (where a CX round corresponds to an ASAP layer without any telegates). Details of

[87] Please refer to [67, Appendix C] for a discussion of the effect of not protecting against such errors.

this can be found in the `DQCode` open-source code[88]. Additionally, similar to the verification circuit construction, we require a fresh ancilla for each measurement to enable higher parallelism of stabiliser measurements (for both verification circuits and measurement-based encoding). The results of the resource comparison across multiple codes and architectures are presented in Section 5.3.

5.2.5 DQC Simulation Design

Having discussed the components of our simulation, from circuits over noise model and decoder to the relevant metrics, we lastly combine them into a unified DQC simulation framework, as visualised in Figure 22. This setup ultimately lets us assess the performance of our fault-tolerant state preparation circuits in a realistic DQC setting.

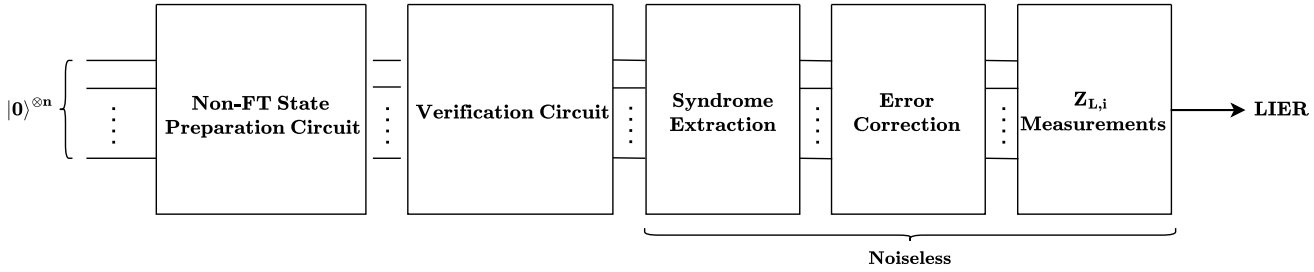


Figure 22: Full DQC simulation workflow to determine the LIER, including non-FT state preparation and verification circuits (cf. Figure 17) as well as noiseless syndrome extraction, error correction and logical-Z operator measurements (cf. Section 5.2.4).

Figure 24 provides an excerpt of the full set of DQC operations simulated by the environment for the Trivariate Bicycle-[[12,2,3]] code on two cores. These operations include the (noisy) non-FT state preparation and verification circuits (cf. Figure 17), compiled to the required telegates via communication qubits, as well as the final (noiseless) syndrome extraction and logical Z-measurement circuits. In particular, we firstly map data qubits onto their assigned QPU cores via (noiseless) SWAP operations[89] to achieve the desired mapping based on the hypergraph partitioning introduced in Section 4.1.3. Then, we apply gate operations on the mapped qubits; for example, the fifth CX gate in Figure 18 has control qubit 5 and target qubit 10, which are mapped to indices 2 and 6 (in the first core), respectively, as indicated in Figure 24. Throughout, we use the communication qubits of the corresponding cores to mediate telegates between cores; in Figure 24, a telegate between data qubits 6 and 11[90] is highlighted[91]. For all error locations in the circuit, noise is sampled stochastically from the depolarsing channel, as introduced in Section 5.2.1. After performing all required state preparation and verification operations, we perform noiseless syndrome extraction and extract our metrics of interest.

Details of the DQC simulation workflow may be inspected in the `DQCode` reposi-

[88] The particular script is accessible at https://github.com/timneumann1/DQCode/blob/main/src/resource_estimate.jl.

[89] More detail on this is provided in Appendix B.20.

[90] Those qubits are mapped to DQC indices 3 and 11, located in the first and second core, respectively.

[91] Note that this telegate is the second two-qubit gate in the raw circuit depicted in Figure 17; however, due to ASAP scheduling, the gate ordering can differ when compiling a circuit for the DQC simulation.

tory[92]. In our code implementation, we build upon the `QuantumClifford` library [35], which exposes an efficient Clifford simulator, Monte Carlo noise sampling and decoding in the `julia` programming language.

Figure 23 shows, as a representative example, the runtime of noisy DQC simulations for the $[[12,2,3]]$ code on two cores, where each data point is the result of averaging three iterations of the entire state preparation and evaluation routine for the corresponding number of samples[93]. As expected, we find a linear overhead[94] of the runtime in the number of samples used.

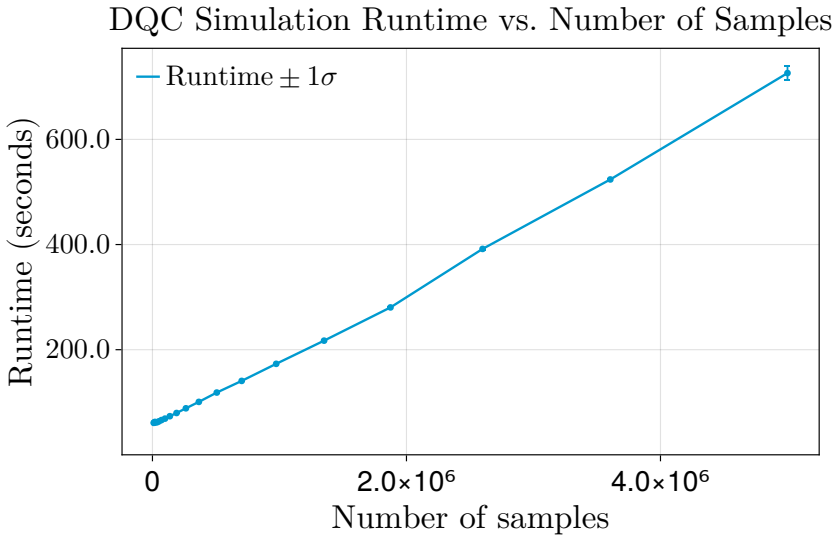


Figure 23: Scaling of DQC simulation runtime with number of samples in `QuantumClifford` Clifford simulation, including noise sampling and extraction of metrics, for Trivariate Bicycle-$[[12,2,3]]$ code on two cores.

[92]In particular, the file https://github.com/timneumann1/DQCode/blob/main/src/dqc_simulator.jl implements the DQC simulation.

[93]The error probabilities used for these runs match the lowest ones in the noise sweep in order to avoid early termination of the run.

[94]The simulation includes a small constant overhead for verification circuit synthesis, which contributes a negligible offset.

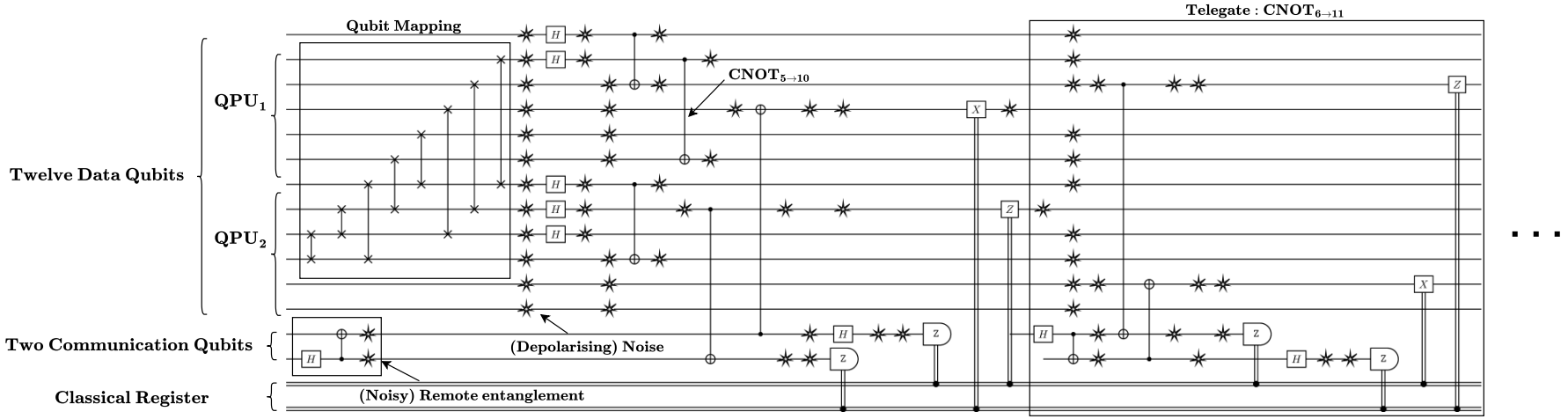


Figure 24: Excerpt of non-FT state preparation circuit from full DQC simulation routine, including mapping of qubits, telegates and noise. Due to the large amount of overall operations, only the first couple of gates from the non-FT state preparation circuit in Figure 17 are displayed. The first $n = 12$ contain the data qubits, on which the code is defined, followed by $n_{\text{cores}}(n_{\text{cores}} - 1) = 2$ communication qubits and ancilla qubits for verification circuit and syndrome extraction (qubit labels are increasing from top to bottom).

5.3 Results & Discussion

Next, we present and discuss our numerical results based on the simulation methods and metrics defined in the previous section. In particular, we will extensively investigate the logical initialisation error rate and its corresponding pseudothresholds, as well as acceptance rates, across multiple codes and architectures; the following discussion of LIER scaling and pseudothresholds herein directly addresses our Research Questions 4 and 5 from Section 3.2. Furthermore, we consider pre-decoding rates and resource estimates. Our focus here lies on the two Multivariate Bicycle codes, with other codes being analysed for reference; the complete dataset is accessible online (cf. Appendix A.1).

LIER Scaling & Pseudothresholds. As discussed in Section 2.1.10, for a distance-d fault-tolerant procedure, we expect the logical error rate to scale as $p_L \sim p^{t+1}$, where p is the physical error rate, and $t = \lfloor (d-1)/2 \rfloor$. Analogously, for our fault-tolerant state preparation routine, we generally expect a scaling of LIER $= Cp^{t+1}$, where p denotes the primary intra-core operation error rate, including the effective physical initialisation error rate $p = p_{\text{init}}$. Figure 25 shows how the LIER scales with p for different code-architecture pairs, depending on the respective Bell state initialisation error rate[95] indicated in the caption. The relation is plotted in log-log space, such that a scaling of the form LIER $= Cp^{t+1}$ translates to $\log(\text{LIER}) = \log(Cp^{t+1}) = \log(C) + (t+1)\log(p)$. Conveniently, assuming that the LIER scales according to some power law, this lets us extract the scaling exponent t from the slope, and the coefficient C from the offset of the resulting linear function in log-log space.

We find that in Figure 25, the slopes and offsets vary, but that most LIERs scale quadratically with p. Furthermore, all simulations yield a LIER below p for some pseudotreshold p_{th}. These observation will be made more precise in Table 4, which captures the scaling coefficients and exponents as well as the pseudothresholds for all simulations.

Next, we zoom in to the specific case of the Trivariate Bicycle code on two cores, which has served as motivating example throughout this work. For this code, Figure 26 visualises the LIER as well as the corresponding acceptance ratios over p. For the smallest considered Bell state initialisation error rate $p_{\text{Bell}} = 9.0 \cdot 10^{-4}$, we perform a least-squares fit[96] of the data to a power law LIER $= Cp^{\alpha}$, in order to identify the coefficient and exponent governing the relation between LIER and p[97]. From this, we estimate the pseudothreshold p_{th}, for which we have $p_{\text{th}} = Cp_{\text{th}}^{\alpha}$, i.e., $p_{\text{th}} = C^{1/(1-\alpha)}$[98], indicated by a vertical line in the figure.

[95] We are visualising the results for the smallest p_{Bell} (which upper-bounds the largest p of the sweep, as discussed in Section 5.2.1) used in the DQC simulation for each code-architecture pair.

[96] The fit is performed using the `LsqFit` library (`https://github.com/JuliaNLSolvers/LsqFit.jl`, Version 0.16.0).

[97] In Definition 10, we stated that the LIER also depends on p_{Bell}; here, we obtain a pseudothreshold conditioned on this remote-entanglement error probability.

[98] While we focus our attention on the exponent for evidence of fault-tolerant scaling, we see that the pseudothreshold as the marker of usefulness for state preparation depends on both the coefficient and the exponent.

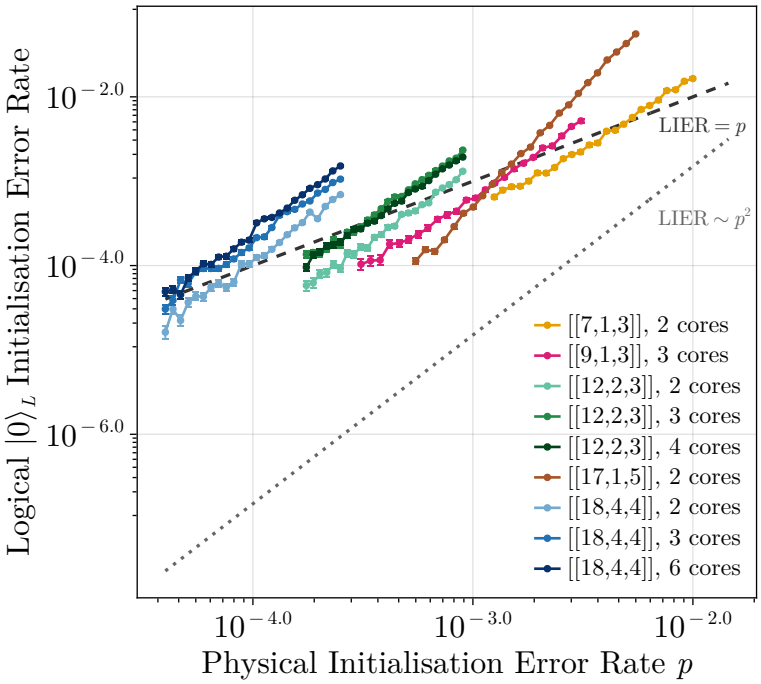


Figure 25: LIER vs. Physical (Initialisation) Error Rate p around the pseudothreshold across multiple code-architecture pairs. Reference lines indicate linear and quadratic power law scaling where the former determines pseudothresholds, and the latter corresponds to the expected scaling for codes of distance $d \in \{3,4\}$. Intra-core noise model as outlined in Section 5.2.1; Bell state initialisation error rates p_{Bell}: 10^{-2} for [[7,1,3]], $10^{-2.5}$ for [[9,1,3]], 10^{-3} for [[12,2,3]], $10^{-2.3}$ for [[17,1,5]], $10^{-3.6}$ for [[18,4,4]].

As Figure 26 indicates, the LIER scales with an exponent of $\alpha = 1.96$ in p for $p_{\text{Bell}} = 9.0 \cdot 10^{-4} = 10^{-3.046}$. Since the given code has distance 3, we expect an exponent of $t+1 = \lfloor (3-1)/2 + 1 \rfloor = 2$, so this result provides numerical evidence for the fault-tolerance of our state preparation routine. Furthermore, we identify a pseudothreshold at $p_{\text{th}} = 6.4 \cdot 10^{-4} = 10^{-3.194}$, which lies in a realistic intra-core noise regime for near-term experiments. And even though a remote-entanglement error of $p_{\text{Bell}} = 9.0 \cdot 10^{-4}$ is more than an order of magnitude lower than what has been achieved in experimental demonstrations, this might not be a hard constraint for near-term systems; our error model is quite conservative, and heavily penalises each of the six telegates in the encoding circuit, e.g., by adding extensive idling noise.

Overall, we observe that p_{Bell} plays a significant role in the scaling, since it impacts both the coefficient C and the exponent α in the power law scaling. Recall that the coefficient roughly corresponds to the number of error locations in some QEC routine, given that errors are independent and of uniform probability, whereas the exponent indicates how many errors would be needed to force a logical error. Firstly, for sufficiently small p_{Bell}, we find the expected scaling $\alpha \approx t+1$, yet with a large coefficient of $C \approx 1198$[99]; this is likely caused by the presence of telegates, which lead to elevated noise levels on the communication qubits, and furthermore induce significant idling times. Secondly, once the strength of p_{Bell} significantly outweighs p, we find that also the exponent α starts to reduce. For $p_{\text{Bell}} = 2.6 \cdot 10^{-3} = 10^{-2.585}$, we still obtain a pseu-

[99] Such large coefficients impact $p_{\text{th}}(C,\alpha)$ negatively (by shifting the pseudothreshold down in p), even though the overall scaling is quadratic.

dothreshold, while for $p_{\text{Bell}} = 8.0 \cdot 10^{-3}$, we encounter a noise floor, such that the LIER stays above p altogether (which directly addresses Research Question 4 in Section 3).

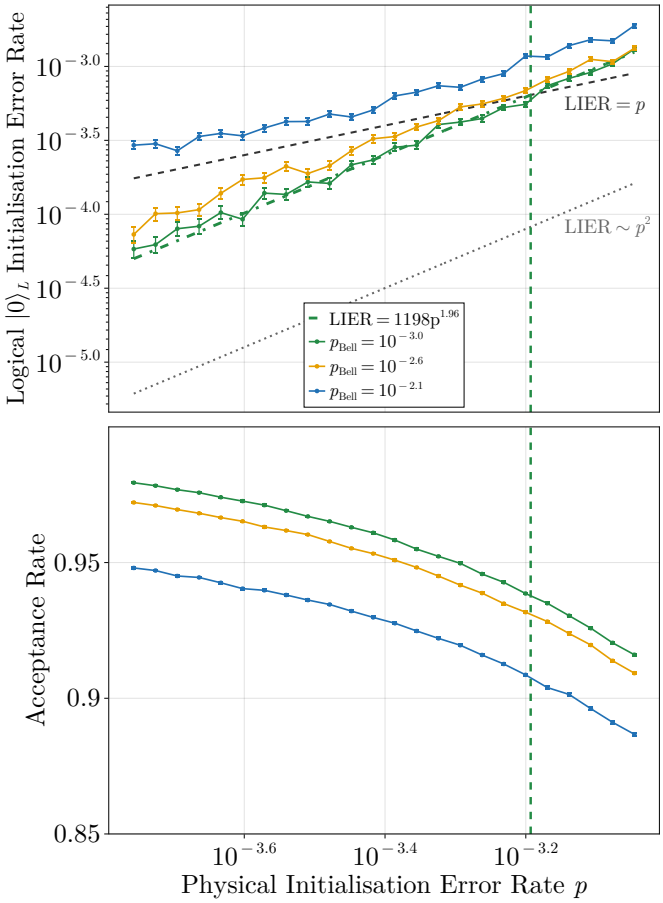


Figure 26: Fault-Tolerant Scaling of LIER vs. Physical Initialisation Error Rate for the Trivariate Bicycle-[[12,2,3] code on two cores. Top: Log-Log Plot relating logical and physical initialisation error rate. Bottom: Acceptance Rate per physical error rate. Power Law fit indicated for smallest p_{Bell}. Standard Deviations for LIER and AR reported according to Bernoulli random variable sample mean variances (cf. Section 5.2.4). Intra-core noise model as outlined in Section 5.2.1.

This behaviour can be explained by recalling our assumption of uniform noise for the fault-tolerance relation LIER $\sim p^{t+1}$; once the error sources are separated too much in terms of error probability, this assumption no longer holds, and the elevated noise does not only increase the scaling coefficient, but also decreases the exponent. Intuitively, the communication qubits 'inject' elevated Bell state initialisation error rates into the system (or onto the ancillas) with every telegate: As long as this error rate is comparable to the error rates at the remaining error locations (and the number of telegates is not prohibitively large), it acts as if there were more error locations, thus primarily increasing the coefficient (or lowering the acceptance rate); however, when at a certain point the elevated noise becomes disproportionately large, it dominates the entirety of noise in the system (leading to average error rates much larger than p), and decreasing p does not change the LIER much anymore.

Hence, our numerical results indicate that the fault-tolerant state preparation relies on the noise sources not to be too strongly separated in strength. To make this

concrete, we investigate the relation between p_{Bell} and p at the pseudothreshold (in view of Research Question 5). Fortunately, as Figure 27 exemplifies, we are capable of tolerating much larger Bell state initialisation error rates around pseudothresholds for many p; in particular, for $p \in [10^{-3.7}, 10^{-3.4}]$, this separation amounts to approximately one order of magnitude[100].

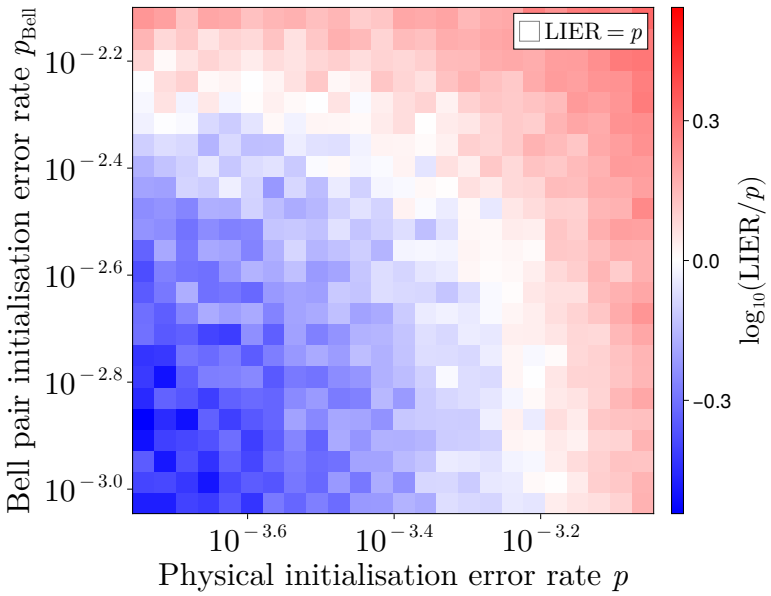


Figure 27: LIER in two-dimensional error space spanned by physical and Bell state initialisation error rates, for the [[12,2,3] code on two cores. Pseudothresholds indicated by bright cells. Intra-core noise model as outlined in Section 5.2.1.

Therefore, although p_{Bell} must not be too large in order to avoid strong noise floors around potential pseudothresholds, its effect can be contained to only impact the coefficient C up to a reasonable separation of the two noise strengths[101]. Assuming that improvements in hardware yield decreased error rates both for intra- and inter-core operations, we may thus reasonably expect the LIER to avoid noise floors and exhibit fault-tolerant scaling as long as the two noise sources are reduced in tandem.

Based on the runs with smallest p_{Bell}, Table 4 reports the LIER scaling results for all codes and architectures. Throughout, we find that the exponent α is close to the expected value of $\lfloor (d-1)/2 \rfloor$, providing further evidence of fault-tolerant state preparation via the distributed verification circuit approach across codes and architectures (thereby addressing Research Question 2). Also, we identify large scaling coefficients C for all instances, which is a result of the large noise associated with telegates, as discussed before. Consequently, the pseudothresholds are relatively small when the number of telegates is large[102].

The acceptance rates for all code-architecture pairs is well above 90% at p_{th}, with the exception of the [[7,1,3]] code (AR ≈ 0.875) and the [[17,1,5]] code (AR ≈ 0.45). For the latter, we hypothesise the main driver of the comparably low AR to be its larger distance, which necessitates checking for a larger amount of harmful errors.

[100] This is similar to the result presented in [131], even though the setting therein is not logical state preparation, but the connection of logical qubit patches.

[101] This behaviour is also supported by our effort to minimise the number of telegates in the circuit.

[102] Here, a direct comparison of pseudothresholds is only possible between different architectures of

QEC Code	Distance d	No. of Cores	No. of Telegates	p_{Bell}	C	α	p_{th} (est.)
[[7,1,3]]	3	2	3	10^{-2}	39.44 ± 6.97	1.68 ± 0.04	$4.6 \cdot 10^{-3}$
[[9,1,3]]	3	3	4	$3.1 \cdot 10^{-3}$	312.94 ± 69.19	1.91 ± 0.04	$1.8 \cdot 10^{-3}$
[[12,2,3]]	3	4	8	$9 \cdot 10^{-4}$	666.73 ± 116.67	1.82 ± 0.02	$3.4 \cdot 10^{-4}$
	3	3	7	$9 \cdot 10^{-4}$	920.87 ± 195.91	1.84 ± 0.03	$3.0 \cdot 10^{-4}$
	3	2	6	$9 \cdot 10^{-4}$	1198.16 ± 397.47	1.96 ± 0.05	$6.4 \cdot 10^{-4}$
[[17,1,5]]	5	2	16	$5.5 \cdot 10^{-3}$	56241.51 ± 10336.19	2.67 ± 0.03	$1.3 \cdot 10^{-3}$
[[18,4,4]]	4	6	35	$2.5 \cdot 10^{-4}$	18087.52 ± 4848.13	1.97 ± 0.03	$3.9 \cdot 10^{-5}$
	4	3	21	$2.5 \cdot 10^{-4}$	3953.34 ± 1365.67	1.82 ± 0.04	$4.2 \cdot 10^{-5}$
	4	2	13	$2.5 \cdot 10^{-4}$	8916.13 ± 5028.25	1.97 ± 0.07	$8.8 \cdot 10^{-5}$

Table 4: Comparison of scaling coefficient C and exponent $\alpha \approx t$ as well as estimated pseudothreshold p_{th} across codes and architectures. Values as extracted from power-law fit based on the data from DQC simulations, and reported with one standard deviation for the lowest p_{Bell} for each simulation run (cf. Figure 25). For fault-tolerant scaling, the exponent α is expected to behave like $\lfloor (d-1)/2 \rfloor$. Number of telegates refers to the combined number of remote gates in both the non-FT state preparation and the verification circuit.

The pseudothresholds for the Trivariate Bicycle code correspond to attainable noise levels for near-term distributed machines, yet rely on advancements in fidelities for remote entanglement[103]. And also the Bivariate Bicycle code exhibits promising fault-tolerant scaling, where the required noise levels for remote entanglement however pose a significant challenge to near-term distributed systems. While the Multivariate Bicycle codes are constructed for relatively large encoding rates, the Steane-[[7,1,3]] and Color-[[17,1,5]] code with $k = 1$ logical qubit on two cores each represent even more promising candidates for immediate demonstrations of logical state preparation on Type-II distributed architectures; here, we find pseudothresholds at $p_{\text{th}} = 4.6 \cdot 10^{-3}$ and $p_{\text{th}} = 1.3 \cdot 10^{-3}$ for $p_{\text{Bell}} = 10^{-2}$ and $p_{\text{Bell}} = 5.5 \cdot 10^{-3}$, respectively[104]. The scaling of the LIER for these codes is visualised in Figure 28.

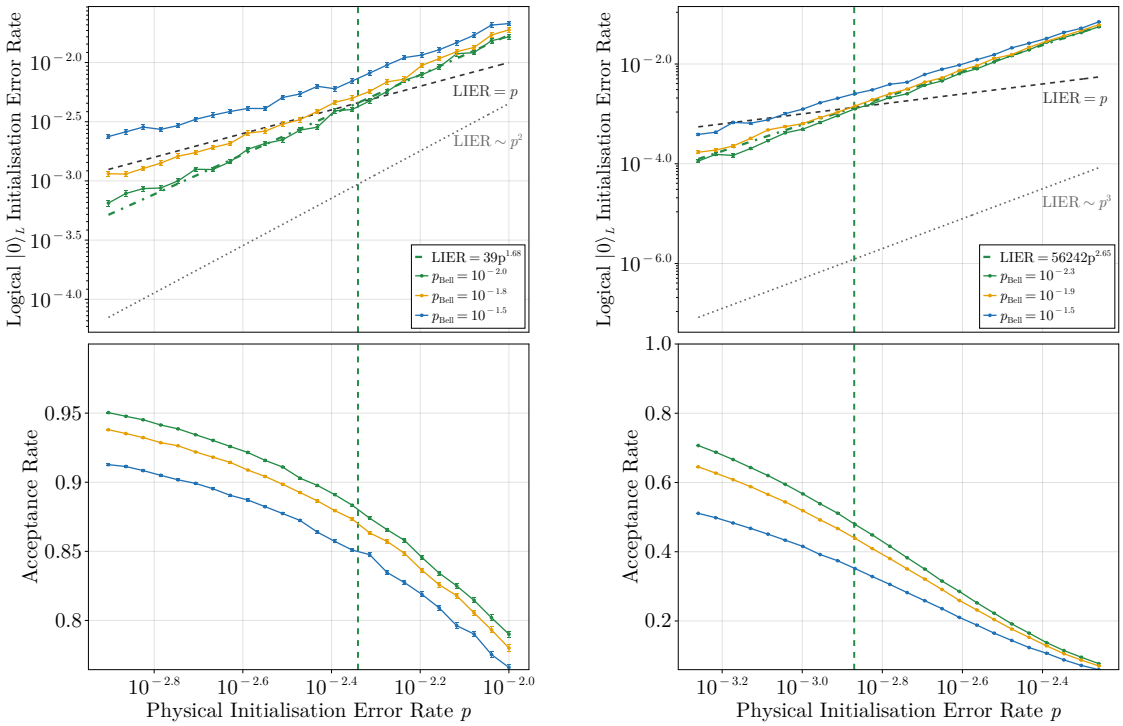
Despite the trade-off between pseudothresholds and encoding rate, in view of Research Question 5 in Section 3, it is reasonable to expect that the required error probabilities for useful proof-of-principle experiment across all codes can be achieved near-term, e.g., on a system of two ion traps with a photonic interconnect. Herein, as Table 4 shows, the number of cores in a Type-II architecture impacts the feasibility of fault-tolerant (qLDPC) state preparation, which is elucidated in the next section.

Number of Cores. In Table 4, we see that scaling parameters and pseudothresholds depend on the number of cores we distribute a given code over. For the Trivariate- and Bivariate Bicycle code, the experiments on two cores yield both the best (largest) pseudothresholds $p_{\text{th}} = 6.4 \cdot 10^{-4}$ and $p_{\text{th}} = 8.8 \cdot 10^{-5}$ as well as the best scaling exponents

each of the Multivariate Bicycle codes (since p_{Bell} varies among codes).

[103] Such advancements are also necessary in order to perform experiments well below the pseudothreshold, without encountering noise floors.

[104] Even though the acceptance rate at the threshold for these codes is lower than for the other codes, the overhead associated with repeating runs is not prohibitively large.



(a) Steane-[[7,1,3]] code on two cores.

(b) Color-[[17,1,5]] code on two cores.

Figure 28: Scaling of LIER vs. physical initialisation error rate for the [[7,1,3]] and [[17,1,5]] codes with $k = 1$ logical qubit, exhibiting fault tolerance favourable for near-term implementations. Intra-core noise model as outlined in Section 5.2.1.

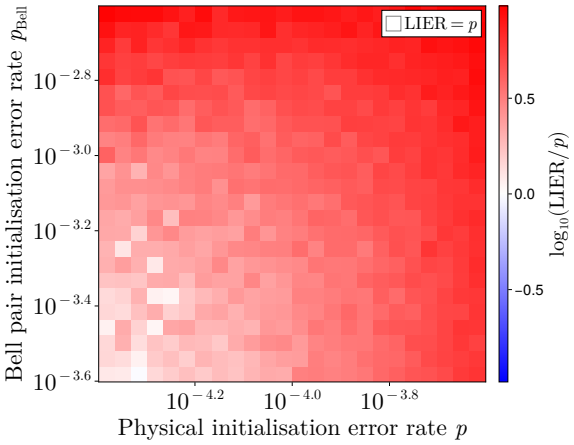
$\alpha = 1.96$ and $\alpha = 1.97$, respectively, across the three architecture sizes each[105]. This is partly a consequence of the optimisation results from Section 4, where using less cores generally led to a reduced number of required telegates[106]. To visualise the impact of the number of cores further, we compare the two-dimensional LIER landscape for the Bivariate Bicycle-[[18,4,4]]-code on two, three and six cores, each simulated for identical noise regimes. Figure 29 shows how the pseudothreshold front moves towards larger p and p_{Bell} when decreasing the number of cores. And even though the pseudothresholds for the 18-qubit code in question rely on very low Bell state initialisation error rates, for the case of two cores with 13 telegates, we can again tolerate much noisier p_{Bell} levels compared to p at the pseudothresholds in the regime $p \in [10^{-4.4}, 10^{-4.1}]$.

Overall, the small instances we investigate here seem to validate the idea that it is generally favourable to make a system as monolithic as possible, shedding light on Research Question 6[107]. This principle may help to guide the transition to a distributed setting, as necessitated by QPU size constraints.

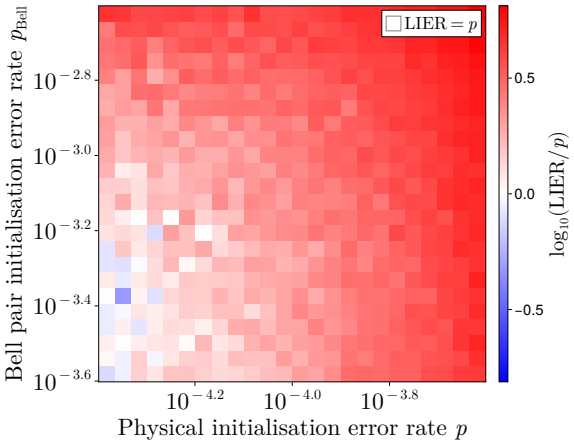
[105] For the Bivariate Bicycle code on six cores, we find a large scaling exponent of $\alpha = 1.97$ as well, which is however also impacted by the fact that the power law fit for this particular code-architecture pair is obtained from data that lies mostly above the pseudothreshold and is therefore biased towards better scaling.

[106] The pseudothresholds for the Trivariate Bicycle code on three and four cores are comparable, most likely due to a small difference in telegate counts.

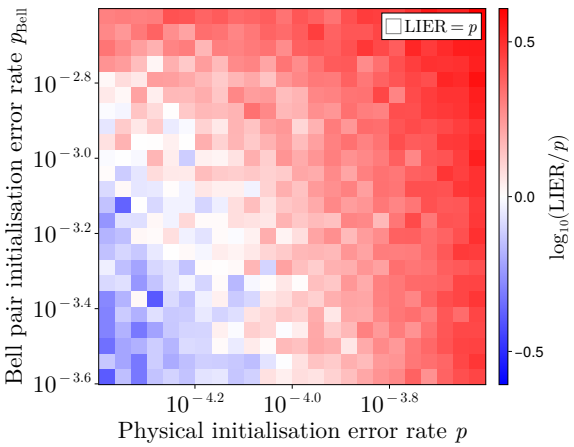
[107] Note that these simulation results strongly on the optimisation outcome, and in particular, the telegate count of the optimised circuits.



(a) Six Cores (35 Telegates)



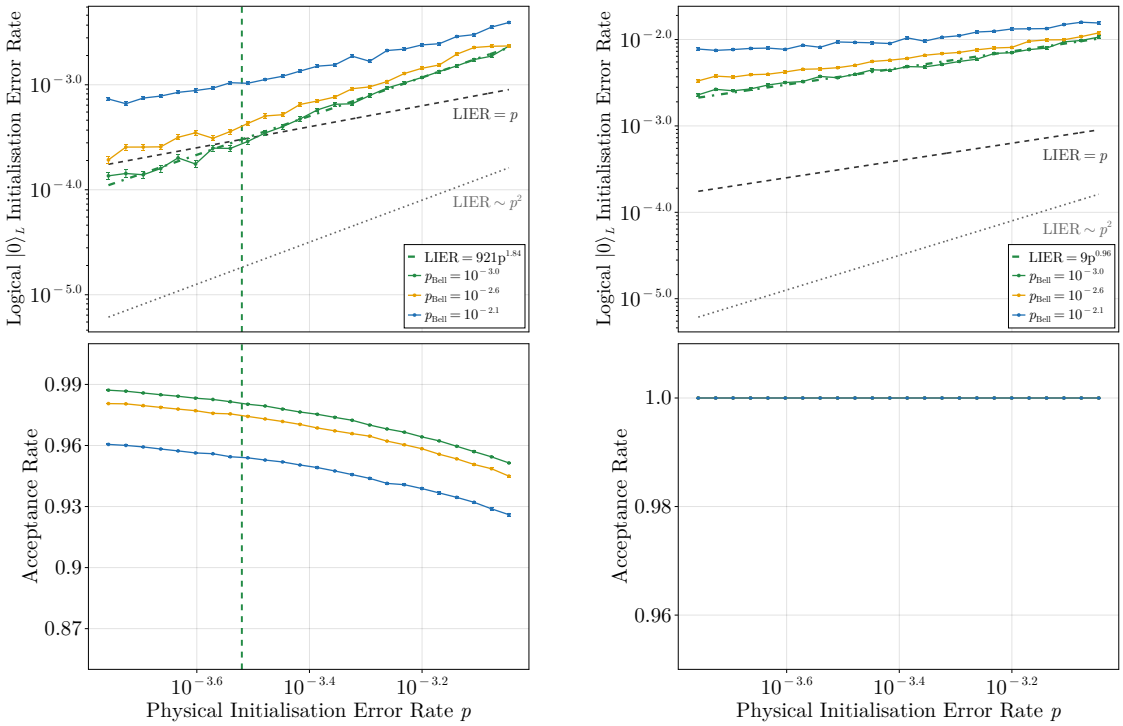
(b) Three Cores (21 Telegates)



(c) Two Cores (13 Telegates)

Figure 29: LIER landscape for the $[[18,4,4]]$ code over various choices of architecture. Intra-core noise model as outlined in Section 5.2.1. a) Six cores with three data qubits each; b) Three cores with six data qubits each; c) Two cores with nine data qubits each. Telegate counts include non-FT state preparation circuit and verification circuit.

FT- vs. Non-FT State Preparation. To clearly point out the effect of the distributed verification circuits for fault-tolerant state preparation, we compare the LIER scaling of optimised non-FT state preparation circuit both with and without an added verification circuit, for the Trivariate Bicycle-$[[12,2,3]]$ code on three cores. Figure 30 shows that the non-FT state preparation routine entirely fails to exhibit a pseudothreshold, with the LIER being constantly much larger than p. This happens since logical errors (or high-weight errors which decoding completes to a logical error) are never discarded. Trivially, the acceptance rate is $AR = 1.0$ without a verification circuit, whereas we also find a large acceptance rate ($\geq 90\%$) in the fault-tolerant setting. Note that p_{th} is shifted towards lower noise levels compared to the same code on two cores, similar to Figure 29.



(a) Fault-tolerant state preparation.

(b) Non-fault-tolerant state preparation.

Figure 30: Scaling of LIER and AR vs. p for Trivariate Bicycle code on three cores. Intra-core noise model as outlined in Section 5.2.1. Left: Fault-tolerant state preparation routine. Right: Non-FT state preparation routine (without verification circuit).

In the context of these numerical results of acceptance rates, we provide some more context on the effect of working with verification circuits. We focus on the left-most data points in the above plots ($p = 10^{-3.757} = 1.75 \cdot 10^{-4}$) and the lowest $p_{\text{Bell}} = 9 \cdot 10^{-4} = 10^{-3.046}$. Firstly, for the fault-tolerant simulation (left plot) of this combination of noise values, we sampled $n_{\text{traj}} = 500,000$ noise patterns in our DQC simulations, out of which 6,413 were discarded (AR $= \frac{500,000-6,413}{500,000} = 0.9872$), with an average of 67 logical $Z_{L,i}$-observable errors[108] on the remaining runs (LIER $= \frac{67}{500,000-6,243} = 1.36 \cdot 10^{-4} = 10^{-3.867}$). In contrast, for the non-FT simulation with the same noise combination in the right panel of Figure 30, we sampled 218,261 runs[109], none of which

[108] Since we have $k = 2$, we are averaging over LIER_1 and LIER_2.

[109] Due to the large error rate, the early-stopping criterion was triggered.

were discarded (AR = 1.0), with an average of 500 logical $Z_{L,i}$-observable errors[110] on all runs (LIER = $\frac{500}{218{,}261} = 2.29 \cdot 10^{-3} = 10^{-2.64}$).

Figure 31 captures all this number crunching, rescaled to 500,000 runs for either simulation, and assuming for simplicity that the noise patterns were identical over the two runs[11]

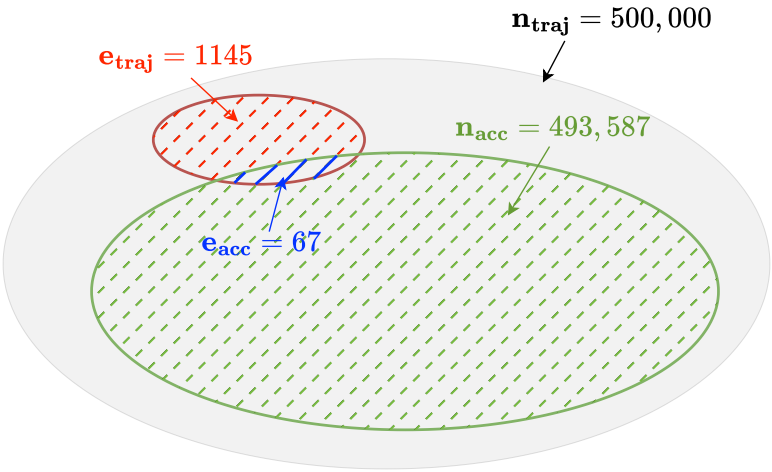


Figure 31: Effect of verification circuit on LIER, based on numbers for FT- and non-FT-state preparation routine of the logical $|00\rangle_L$ state of the $[[12,2,3]]$ code on three cores. Areas of shapes enlarged for visibility and not to proportion.

The main takeaway here is the following: Discarding a small number of runs (a little more than one percent) allows us to reduce the total number of logical errors by over 94% from $e_{\text{traj}} = 1145$[112] to a mere $e_{\text{acc}} \approx 67$ errors among the accepted runs.

While we are generally interested in both minimising e_{acc} at the intersection of all n_{traj} and the e_{traj} faulty runs, and maximising n_{acc}, it is possible that a verification circuit leads us to discard some runs that would have not resulted in an error (e.g., if some verification ancilla is erroneously flipped before measurement, even though no error was present in the system). But given the large AR, it is reasonable to expect that most of the discarded runs in the above example were actually faulty, and thus would have increased the LIER. Therefore, the cost of discarding runs here is comparably small (yielding a large AR), while resulting in a significant reduction in LIER.

[110] In this case, LIER_1 and LIER_2 differed strongly, as will be discussed later, but for the overall LIER we are only concerned with the average.

[111] Since sampling is probabilistic, we cannot claim that the underlying noise patterns are precisely the same, but it is reasonable to expect that extreme behaviour averages out over half a million samples.

[112] Here, we rescaled $500 \cdot \frac{500{,}000}{218{,}261} \approx 1145$ errors for the non-FT state preparation simulation.

LIER per Logical Qubit. When defining the LIER as the average over the k individual LIER_i terms (cf. Definition 9), we are not making any claim about the similarity of these initialisation error rates per logical qubit. In Figure 32, we firstly compare the four individual error rates for the $[[18,4,4]]$ code on two cores, in addition to the fidelity F (the 'no-logical-error-rate'). As expected, the fidelity lower bounds the logical initialisation error rates LIER_i, which are reasonably close in magnitude.

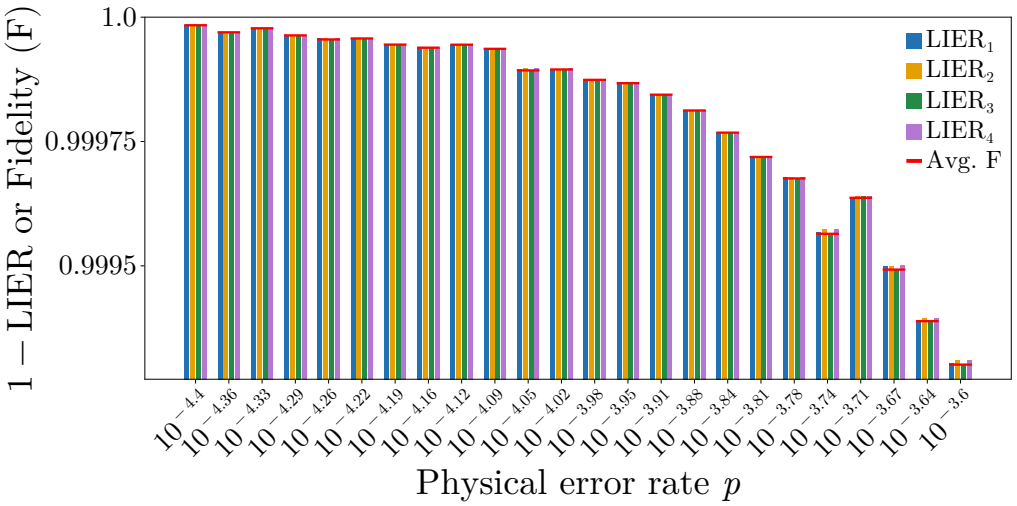
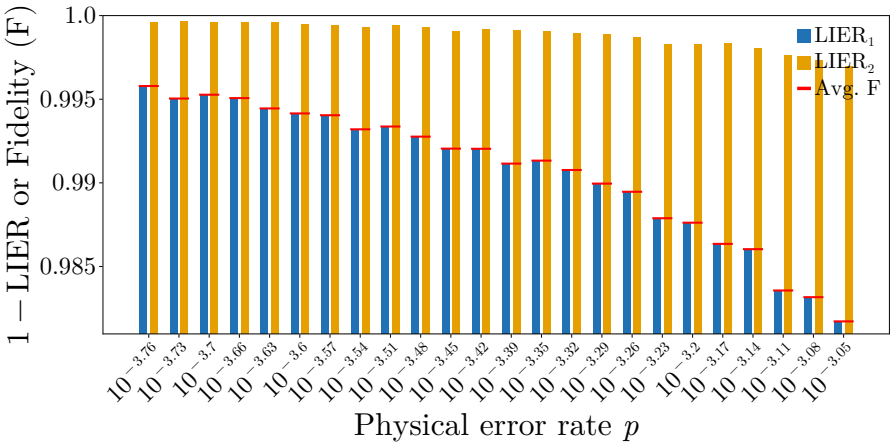


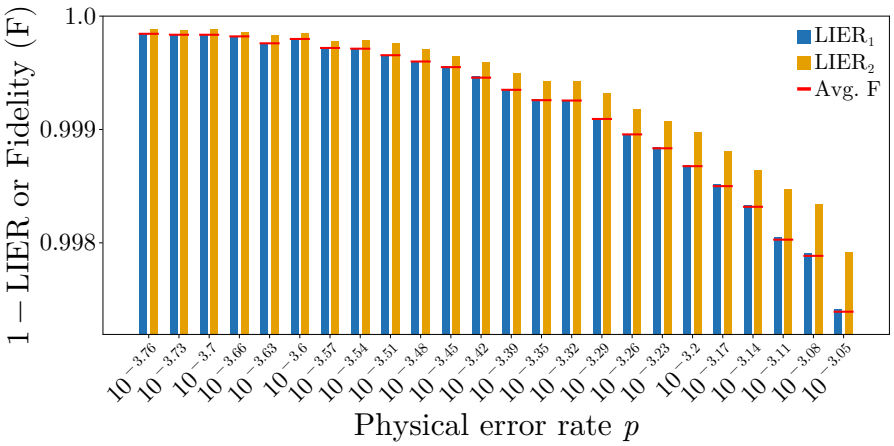
Figure 32: Comparison of $1-\mathrm{LIER}$ and Fidelity F over p across the $k=4$ logical qubits of the $[[18,4,4]]$ Bivariate Bicycle code on two cores. Intra-core noise model as outlined in Section 5.2.1, $p_{\mathrm{Bell}} = 2.5\cdot 10^{-4}$. Fidelity is reported as average over all runs for (p, p_{Bell}).

Interestingly, for the non-FT state preparation circuit of the $[[12,2,3]]$ code we analysed in the previous section, the individual logical initialisation error rates vary significantly, as Figure 33 shows. In particular, the second logical qubit is quite robust, whereas almost all infidelity in logical initialisation stems from the first logical qubit. In the fault-tolerant setting (i.e., including a verification circuit), this discrepancy is reduced, and LIER_1 and LIER_2 are comparable in magnitude with low overall LIER_i levels.

One potential explanation for this behaviour is that depending on the circuit structure of the non-FT state preparation circuit, certain logical X- or Y-errors (or high-weight errors) are more likely to occur. Then, if harmful errors are never caught, the amount of such logical errors might disproportionately affect the first logical qubit. On the other hand, if harmful errors are discarded, we are mitigating the effect of these high-weight errors, and the logical qubits perform comparably again. The fact that the LIER_2 is very similar in both plots, whereas the LIER_1 experiences a strong increase in the fault-tolerant setting (yet still not matching LIER_2 fully), suggests that the verification indeed discards errors particularly harmful to the first logical qubit.



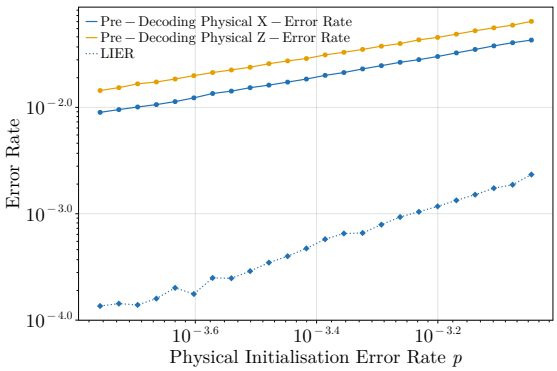
(a) Non-fault-tolerant state preparation.



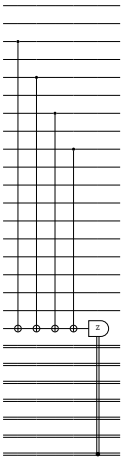
(b) Fault-tolerant state preparation.

Figure 33: Comparison of LIER_1 and LIER_2 for the Trivariate Bicycle-$[[12,2,3]]$ code on three cores. Intra-core noise model as outlined in Section 5.2.1, $p_{\text{Bell}} = 9 \cdot 10^{-4}$. a) Significant difference among logical qubits for non-FT encoding circuit; b) Discrepancy is lifted after the verification circuit is added.

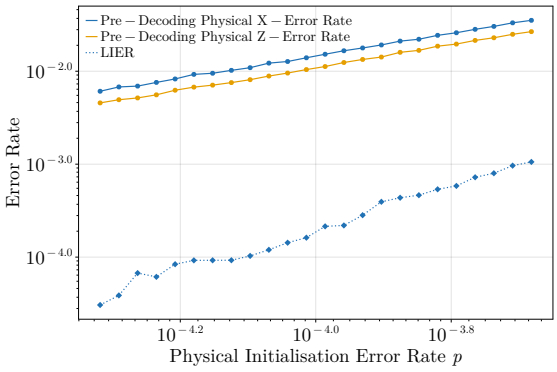
Pre-Decoding Error Rates. Within the paradigm of fault-tolerant state preparation, we already discussed the relevance of protecting CSS codes against both X- and Z-errors. As introduced in Section 5.2.4, we provide a proxy for the relation between these two error types by considering the respective pre-decoding error rates. Figure 34 shows the pre-decoding X- and Z- error rates as well as the logical X-error rate (LIER) for the Trivariate Bicycle-[[12,2,3]] (upper plots) and Bivariate Bicycle-[[18,4,4]] code (lower plots) on three cores. In both cases, the correction of pre-decoding X-errors results in a strongly suppressed logical X-error rate.



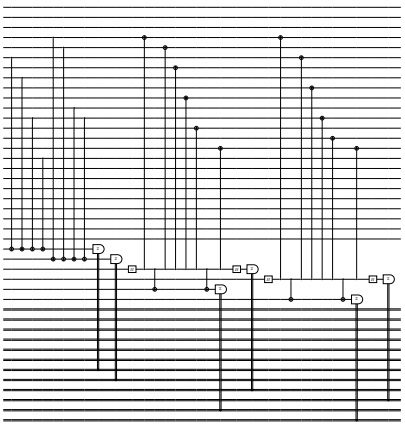
(a) Pre-decoding and logical error rates vs. p for the Trivariate Bicycle [[12,2,3]] code on three cores.



(b) Verification circuit for the [[12,2,3]] code on three cores.



(c) Pre-decoding and logical error rates vs. p for the Bivariate Bicycle [[18,4,4]] code on three cores.



(d) Verification circuit for the [[18,4,4]] code on three cores.

Figure 34: Pre-Decoding error rates and verification circuit structure. Top: Trivariate Bicycle-[[12,2,3]] code on three cores (intra-core noise model as outlined in Section 5.2.1, $p_{\text{Bell}} = 9 \cdot 10^{-4}$). Bottom: Bivariate Bicycle-[[18,4,4]] code on three cores (intra-core noise model as outlined in Section 5.2.1, $p_{\text{Bell}} = 2.5 \cdot 10^{-4}$).

For the Trivariate Bicycle code, we find that the pre-decoding Z-error rate is slightly larger than the respective X-error rate. Inspecting the verification circuit synthesised for this code-architecture pair, we see that it consists of a single Z-type parity check (catching harmful X-errors). This can be a reason for the elevated Z-type error: Z-errors can propagate from the ancilla onto the data qubits, and even though they are not harmful (otherwise there would be a X-type verification afterwards), these raw

errors contribute to a higher Z-error pre-decoding error rate. Across all codes and architectures, we find that a sole Z-type verification circuit results in slightly elevated Z-type pre-decoding errors.

On the other hand, if the Z-type verification circuit propagates harmful errors, such that a X-type verification circuit needs to be appended[113], the specific circuit structure might favour raw X- or Z-type errors. For example, the verification circuit for the $[[18,4,4]]$ code on three cores consists of both Z- and X-type parity checks, and yields a larger X-type pre-decoding error rate, as Figure 34 shows.

Since logical Z-operators can be utilised as verification stabilisers, we expect a small bias towards an elevated Z-error pre-decoding error rate. Overall, however, we find that the pre-decoding error rates of the two different error types are comparable in magnitude, with the specific (verification) circuit structure determining which source dominates. We wish to stress again that, while harmful errors are discarded by construction, this study acts as a proxy for our understanding of how different errors propagate, but does not constitute a direct investigation of logical Z-error rates.

Resource Comparison. As last metric of interest, we investigate how our fault-tolerant state preparation routine compares to a fully measurement-based approach. As discussed in Section 5.2.4, we assume equivalent hardware and gate scheduling conditions, and assess the performance of the distributed measurement of X-type stabilisers for only one round. Figure 35 provides an example of such comparison for the Trivariate Bicycle-$[[12,2,3]]$ code on three cores; here, we find that the circuit approach requires less gates (and in particular, less telegates), and most importantly exhibits significantly fewer measurements and ancilla qubits[114]. At the same time, the depth of the encoding, both in terms of CX rounds (ASAP layers without any telegate) and telegate rounds is larger.

We wish to stress the importance of the reduced ancilla qubit count in this analysis: Whereas the state preparation circuit only requires one ancilla qubit[115], the distributed stabiliser measurement approach draws upon six ancillas (for the measurement of the six X-type stabilisers). Now, performing some QEC computation on a distributed system, we will inevitably need to utilise distributed stabiliser measurements (both of X- and Z-type) for syndrome extraction. The central takeaway of this analysis is then that our encoding approach is often more efficient in terms of gate operations, while using the required infrastructure for later QEC cycles, i.e., ancilla qubits for verification can later be reused as syndrome extraction ancillas. Thus, performing our fault-tolerant state preparation routine does not create additional overheads in the number of qubits, and even reduces quantum resources compared to measurement-based initialisation.

Table 5 contains a resource comparison between the two approaches for all codes discussed in this work[116]. We find that our approach outperforms the measurement-based encoding for all code-architecture pairs (besides the $[[17,1,5]]$ code on two cores),

[113] Recall that instead of flagging the Z-type verification, we catch such errors with a X-type verification circuit.

[114] These improvements come at the cost of a slightly reduced acceptance ratio, which however does not change the overall comparison qualitatively in the noise regimes of interest.

[115] This ancilla is placed in the third core due to its interaction with other qubits.

[116] Note that we merged the columns counting the number of measurements, and counting the number of ancillas, since for both the verification circuit and the measurement-based approach, we assume that every ancilla is measured precisely once. Also, we only report telegate counts and depth, since such gates pose the largest overhead for state preparation.

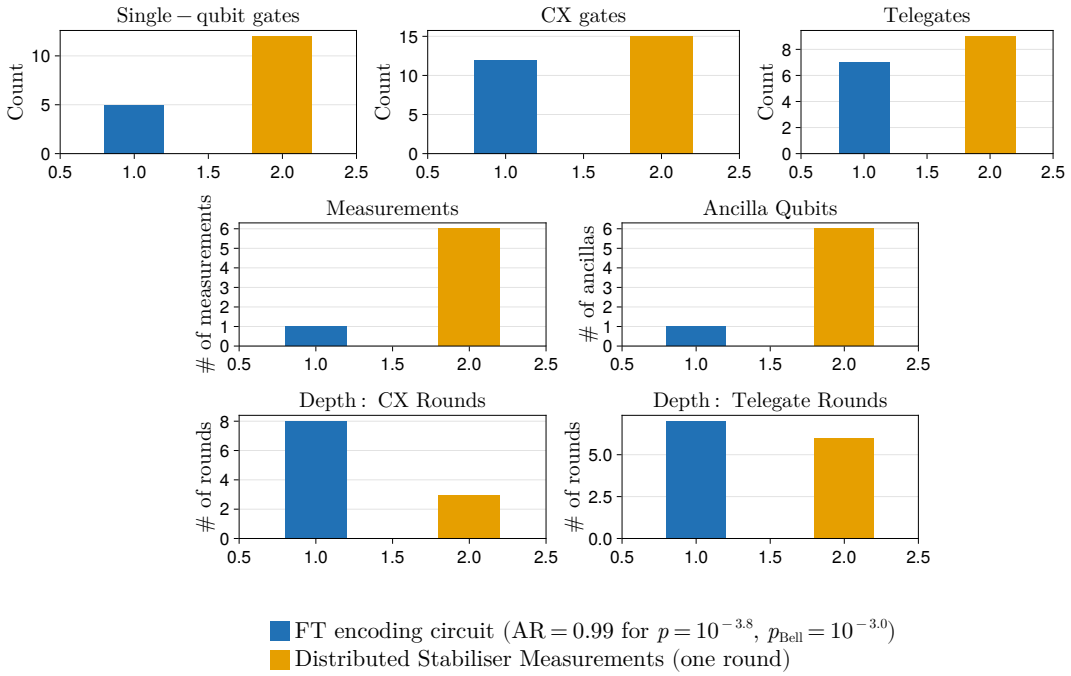


Figure 35: Resource Comparison for $|0\rangle_L^{\otimes k}$ preparation between (i) fault-tolerant state preparation circuit with post-selection and (ii) one round of distributed stabiliser measurements, for the Trivariate Bicycle-[[12,2,3]] code on three cores.

in terms of the number of ancillas (and measurements), while performing comparable or better in terms of telegates, especially if the number of cores is two or three. While our focus lies on the Multivariate Biycycle codes with high encoding rate, the color code is a special case: Since it has distance 5, the verification circuit required for fault-tolerant state preparation is large[117].

While the QEC cycle-aware mapping helps to reduce the amount of non-local operations in distributed stabiliser measurements (e.g., for syndrome extraction), we find that our optimisation of logical state preparation as particular application leads to a low resource overhead on Type-II systems. In view of Research Question 7 from Section 3.2, we expect this effect to be particularly pronounced once we compare with fault-tolerant measurement-based initialisation methods (requiring multiple rounds[118] of distributed stabiliser measurements in general). Furthermore, this comparison settles the third research question: Since our method still relies on measurements in the verification circuit[119], it is not entirely measurement-free; however, given our hardware and scheduling assumptions, it significantly reduces the number of measurements in comparison to a fully measurement-based approach.

[117] We already saw that this leads to a lower acceptance rate, compared to the other codes, which further increases the resource needs for our post-selection approach. Importantly, assuming that we perform $d = 5$ rounds of distributed stabiliser measurements for fault tolerance, our approach still outperforms the measurement-based method in terms of telegates and depth; however, the increased ancilla need for verification remains.

[118] 'Rounds' here refers to the number of times all stabilisers need to be measured, and not to the number of rounds we define in our scheduling procedure in Section 5.2.2.

[119] Note that these measurements could potentially be circumvented by employing a conditional correction method similar to [86].

QEC Code	# Cores	# Telegates	# Measurements/Ancillas	Depth (Telegate Rounds)
[[7,1,3]]	2	**3**|4	**1**|3	**3**|4
[[9,1,3]]	2	**4**|6	**1**|2	**4**|6
[[12,2,3]]	4	**8**|12	**1**|6	**6**|8
	3	**7**|9	**1**|6	7|**6**
	2	6|6	**4**|6	6|6
[[17,1,5]]	2	16|**6**	14|**8**	16|**6**
[[18,4,4]]	6	35|**32**	**6**|9	17|17
	3	**21**|27	**6**|9	**16**|17
	2	**13**|19	**6**|9	**13**|19

Table 5: Resource Comparison for encoding methods across codes in terms of telegates, number of measurements and ancilla qubits, and depth (telegate rounds). Each cell reports the respective metric for the FT state preparation circuit (left entry) and one layer of the distributed stabiliser measurement approach (right entry). Results do not take into account the acceptance ratios, which are ≥ 85% in the noise regime of interest for all codes besides the [[17,1,5]] code.

5.4 Limitations & Future Work

In the previous section, we assessed the performance of our fault-tolerant state preparation routines in a realistic DQC setting. In the following, we provide some limitations of our approach, and outline potential routes for future work.

Firstly, our noise model is inspired by real hardware specifications, hence we generally expect our DQC simulation framework to be a realistic testbed for near-term Type-II systems. However, the results presented here strongly depend on the hardware and scheduling assumptions as well as the choice of error model, including the noise ratios. Therefore, all LIER and noise thresholds are, as usual, to be understood conditioned on these choices. We identified pseudothresholds and a favourable scaling of the LIER with respect to p; however, the existence of noise floors in the encoding of Multivariate Bicycle codes for state-of-the-art values of p_{Bell} indicates the need for hardware advances to yield near-term proof-of-concept experiments for such state preparation. In particular, even though we used a conservative error model, the usefulness of our approach greatly depends on reliable remote connections between QPUs. It would be fascinating to see how our encoding procedure performs in a real system, once such machines exists[120].

Secondly, we simulated verification circuits in a DQC context, yet we did not optimise these circuits specifically for this setting. However, in the `MQT-QECC` library, the (monolithic) gate-optimal synthesis of these circuits already reduces the overhead significantly, which makes our DQC-agnostic adaptation acceptable. The synthesis of verification circuits might also pose a constraint on the scalability of our approach, similar to what we discussed in Section 4. Future work could also investigate alternative methods of verifying state preparation, such as Steane-type Error Detection [165].

In terms of scalability, it seems worthwhile to explore codes of larger distance with lower encoding rate in order to probe error-corrected quantum computation on Type-II systems in the near future. This comes with a couple of challenges, including the

[120]The non-existence of distributed systems akin to the ones we have used as hardware motivation in this work is both a limitation and an opportunity; we cannot yet validate our simulations with real hardware, but we can motivate experimental design with our DQC simulation.

optimisation of non-FT encoding circuits (cf. Section 4 for potential solutions), the synthesis of efficient verification circuits for large distance, and a need to transition away from CSS Lookup Table Decoders to more advanced decoders[121]. While these improvements were out of scope for this work, it would then be interesting to see how the acceptance rate of a state preparation routine scales with increasing distance.

Regarding fault tolerance, we evaluated our encoding circuits on a noise model with two strongly separated noise levels, which the (count-based) fault tolerance design is not directly tailored towards. This contributes to the development of noise floors once the mismatch in error rates becomes too large. Nonetheless, our assessment shows that the fault tolerance design principle provides a good proxy to control the LIER below its physical pendant, especially when the telegate count is limited. Herein, limiting the number of cores according to our results can help to reduce the remote gate overhead for larger codes[122]. An interesting extension to our work would be an assessment of the DQC performance of the circuits under a different noise model; for example, one could investigate which noise processes (dephasing, relaxation, etc.) are particularly relevant on a given hardware, and adapt our platform-agnostic error model to this hardware. We omitted such hardware-tailored noise modeling in order to maintain a certain level of generality.

Besides from investigating higher-distance codes, a promising improvement to the present work lies in the optimisation of the scheduling methods. Whereas we optimised for gate count, and then performed operations based on an ASAP scheduler, combined sequentially with a communication qubit conflict resolution for telegates, we expect there to be more efficient ways to execute a circuit faithfully for a given network; this could involve a forward-looking heuristic or a combined scheduler, instead of our sequential greedy approach. Such optimisation tasks go hand in hand with the underlying hardware assumptions, which might also be adapted to account for heterogeneous network structures, a certain native gate set, or the inclusion of quantum state teleportation, for example.

To further assess the performance of our encoding as a building block in a larger computation, more elaborate methods for determining a proxy for the logical Z-error rate should be analysed. And even for the given method of assessing pre-decoding errors, it could be worthwhile to probe the effect of switching the X- and Z- verification circuits. In the larger context of QEC, our fault-tolerant state preparation routine should be combined with logical operations[123]; this would also contribute to determining the extent to which Type-II systems are useful for near-term scalable quantum computing.

[121] In this context, it might be worthwhile to investigate decoders that handle asymmetric noise.

[122] This in turn also requires a more scalable approach to the optimisation of the circuits.

[123] Within the paradigm of qLDPC codes on Type-II architectures, it will be interesting to see whether logical operations can benefit from a tailored optimisation as well.

6 Conclusion & Outlook

In this work, we introduced `DQCode` as a pipeline for logical $|0\rangle_L^{\otimes k}$ state preparation of the small Trivariate Bicycle-$[[12,2,3]]$ and Bivariate Bicycle-$[[18,4,4]]$ codes on Type-II architectures, at the intersection of qLDPC codes and distributed quantum systems as potential pathway to scalable fault-tolerant quantum computing. In the optimisation stage, we investigated encoding circuits that minimise the number of remote gate teleportations between QPUs, followed by the simulation stage, in which we assessed the performance of these optimised circuits, equipped with verification circuits for fault tolerance, on a realistic DQC noise model. To conclude, in the following we summarise the most relevant results as well as research opportunities originating from this work.

In Section 4, we compared a warm-started Genetic Algorithm and Monte Carlo Tree Search optimisation algorithm with a number of baseline methods. Based on a QEC cycle-aware initial mapping, we found a significant improvement in telegate count for circuits discovered by the GA compared to simple baseline methods (e.g., a $\sim 60\%$ reduction with respect to the monolithic `Gottesman` encoding circuit), and comparable performance with advanced baselines (such as a non-fault-tolerant e-bit compilation approach). Notably, our methods incorporate fault tolerance as design principle into the optimisation. While these results are promising for the small code instances involving up to 18 physical qubits, we encountered scalability of our methods as a bottleneck preventing the analysis of larger codes. This inspires further research into the optimisation of encoding circuits for qLDPC codes of larger distance on Type-II architectures.

In Section 5, we showed that verification circuits can be leveraged to fault-tolerantly encode the logical zero state in a DQC setting. Sweeping error rates — motivated by ratios drawing upon real hardware values — we provided numerical evidence for this claim by identifying LIER scaling exponents matching the theoretical expectation of $t = \lfloor (d-1)/2 \rfloor$ for distance d- codes under suitable conditions. However, once the separation of error probabilities for intra- and inter-core operations in our noise model grew too large, we found this scaling to be limited by noise floors.

Furthermore, we identified noise pseudothresholds below which the LIER outperformed the physical $|0\rangle$ state initialisation error rate p, satisfying a necessary condition for the usefulness of QEC on distributed architectures. Importantly, all experiments tolerated larger Bell state initialisation error rates at these thresholds (with p_{Bell} being elevated approximately one order of magnitude compared to the intra-core error rate p for suitable ranges of p), while maintaining high acceptance rates. For the Trivariate and Bivariate Bicycle code under consideration and Bell state initialisation errors on the order of 10^{-3} and 10^{-4}, respectively, we found pseudothresholds on the order of $p_{\text{th}} = 10^{-4}$ and $p_{\text{th}} = 10^{-5}$, respectively, where p_{th} captures the error probability of a broader class of intra-core operations.

Importantly, under the given hardware assumptions, our fault-tolerant state preparation was shown to require reduced resource needs compared to its alternative of distributed stabiliser measurements. In particular, while repeated QEC operations (such as syndrome extraction) generally benefit from our QEC cycle-aware mapping, our state preparation method allows for a usage of ancillary qubits allocated to such recurring operations without additional overhead. By comparing architectures of different

sizes, we identified a general trend of increased encoding viability for systems with a smaller number of cores, which is a consequence of reduced telegate overhead. This can serve as a design principle, assuming that QPU size constraints necessitate a transition to distributed systems.

Despite the need for improvements in remote-entanglement fidelities to suppress noise floors, the results presented herein indicate the near-term viability of qLDPC code state preparation via fault-tolerant encoding circuits on Type-II architectures. In particular, in comparison with the specifications of modern candidate hardware platforms, we identify a potential for logical state preparation experiments, especially of the Trivariate Bicycle code, on small near-term distributed systems such as ion traps with photonic interconnects. And whereas `DQCode` extends existing state preparation approaches on monolithic devices by investigating the interplay of qLDPC codes and Type-II systems for fault-tolerant state preparation, future research should include improved scheduling procedures to further close the gap to experimental viability.

References

[1] Thorsten Koch et al. “The Quantum Optimization Benchmarking Library”. In: *Nature Computational Science* 6.6 (2026), pp. 653–671. ISSN: 2662-8457. DOI: 10.1038/s43588-026-00991-1. URL: http://dx.doi.org/10.1038/s43588-026-00991-1.

[2] P.W. Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

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

[4] John Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: http://dx.doi.org/10.22331/q-2018-08-06-79.

[5] Anthony Ransford et al. “A 98-qubit trapped-ion quantum computer with all-to-all connectivity”. In: *Nature* (June 17, 2026). ISSN: 1476-4687. DOI: 10.1038/s41586-026-10676-4. URL: https://doi.org/10.1038/s41586-026-10676-4.

[6] Barbara M. Terhal. “Quantum error correction for quantum memories”. In: *Reviews of Modern Physics* 87.2 (Apr. 2015), pp. 307–346. ISSN: 1539-0756. DOI: 10.1103/revmodphys.87.307.

[7] C. Monroe et al. “Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects”. In: *Physical Review A* 89.2 (Feb. 2014). ISSN: 1094-1622. DOI: 10.1103/physreva.89.022317. URL: http://dx.doi.org/10.1103/PhysRevA.89.022317.

[8] Daniel Gottesman. “Surviving as a Quantum Computer in a Classical World”. 2024. URL: https://www.cs.umd.edu/class/spring2024/cmsc858G/QECCbook-2024-ch1-15.pdf.

[9] Daniel J. Spencer et al. *Quantum error correction and fault tolerance: A comprehensive tutorial*. 2026. arXiv: 2605.29137 [quant-ph]. URL: https://arxiv.org/abs/2605.29137.

[10] W. Wootters and W Zurek. “A single quantum cannot be cloned”. In: *Nature* 299 (1982), pp. 802–803. DOI: https://doi.org/10.1038/299802a0.

[11] R. W. Hamming. “Error detecting and error correcting codes”. In: *The Bell System Technical Journal* 29.2 (1950), pp. 147–160. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[12] C. E. Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[13] P. W. Shor. “Scheme for reducing decoherence in quantum computer memory”. In: *Phys. Rev. A* 52.4 (1995), pp. 2493–2496. DOI: 10.1103/PhysRevA.52.R2493.

[14] Andrew Steane. “Multiple-particle interference and quantum error correction”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 452.1954 (Nov. 1996), pp. 2551–2577. ISSN: 1364-5021. DOI: 10.1098/rspa.1996.0136. URL: https://doi.org/10.1098/rspa.1996.0136.

[15] P. W. Shor and A. R. Calderbank. “Good quantum error-correcting codes exist”. In: *Physical Review A (Atomic, Molecular, and Optical Physics)* 54.2 (1996), pp. 1098–1105. DOI: 10.1103/PhysRevA.54.1098.

[16] A. M. Steane. “Multiple-Particle Interference and Quantum Error Correction”. In: *Proceedings: Mathematical, Physical and Engineering Sciences* 452.1954 (1996), pp. 2551–2577. DOI: 10.1098/rspa.1996.0136.

[17] Daniel Gottesman. “Stabilizer Codes and Quantum Error Correction”. PhD thesis. Pasadena, California: California Institute of Technology, May 21, 2024. URL: https://arxiv.org/pdf/quant-ph/9705052 (visited on 01/23/2026).

[18] A. R. Calderbank et al. *Quantum Error Correction via Codes over GF(4)*. Sept. 10, 1997. DOI: 10.48550/arXiv.quant-ph/9608006. arXiv: quant-ph/9608006.

[19] Sergey Bravyi et al. “High-threshold and low-overhead fault-tolerant quantum memory”. In: *Nature* 627.8005 (Mar. 28, 2024), pp. 778–782. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-024-07107-7. arXiv: 2308.07915[quant-ph]. URL: http://arxiv.org/abs/2308.07915 (visited on 02/02/2026).

[20] John Wright and Uma Girish. *Quantum Coding Theory, Lecture 8: CSS Codes*. Lecture Notes, CS294, UC Berkeley. Feb. 2024. URL: https://people.eecs.berkeley.edu/~jswright/quantumcodingtheory24/scribe%20notes/lecture08.pdf (visited on 08/17/2026).

[21] Raymond Hill. *A First Course in Coding Theory*. Oxford Applied Linguistics. Clarendon Press, 1986. ISBN: 9780198538035.

[22] Oscar Higgott. *PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching*. 2021. arXiv: 2105.13082 [quant-ph]. URL: https://arxiv.org/abs/2105.13082.

[23] Oscar Higgott and Craig Gidney. “Sparse Blossom: correcting a million errors per core second with minimum-weight matching”. In: *Quantum* 9 (Jan. 2025), p. 1600. ISSN: 2521-327X. DOI: 10.22331/q-2025-01-20-1600. URL: https://doi.org/10.22331/q-2025-01-20-1600.

[24] Neereja Sundaresan et al. “Demonstrating multi-round subsystem quantum error correction using matching and maximum likelihood decoders”. In: *Nature Communications* 14.1 (May 2023). ISSN: 2041-1723. DOI: 10.1038/s41467-023-38247-5. URL: http://dx.doi.org/10.1038/s41467-023-38247-5.

[25] Hanwen Yao et al. *Belief Propagation Decoding of Quantum LDPC Codes with Guided Decimation*. 2024. arXiv: 2312.10950 [cs.IT]. URL: https://arxiv.org/abs/2312.10950.

[26] Remmy Zen et al. “Quantum Circuit Discovery for Fault-Tolerant Logical State Preparation with Reinforcement Learning”. In: *Physical Review X* 15.4 (Oct. 22, 2025), p. 041012. ISSN: 2160-3308. DOI: 10.1103/gqpr-dgz7.

[27] Tom Peham et al. “Automated Synthesis of Fault-Tolerant State Preparation Circuits for Quantum Error-Correction Codes”. In: *PRX Quantum* 6.2 (May 2025). ISSN: 2691-3399. DOI: 10.1103/prxquantum.6.020330. URL: http://dx.doi.org/10.1103/PRXQuantum.6.020330.

[28] Yifan Hong. “Single–shot preparation of hypergraph product codes via dimension jump”. In: *Quantum* 9 (Oct. 2025), p. 1879. ISSN: 2521-327X. DOI: 10.22331/q-2025-10-07-1879. URL: http://dx.doi.org/10.22331/q-2025-10-07-1879.

[29] Daniel Gottesman. *The Heisenberg Representation of Quantum Computers*. July 1, 1998. DOI: 10.48550/arXiv.quant-ph/9807006. arXiv: quant-ph/9807006.

[30] Scott Aaronson and Daniel Gottesman. “Improved Simulation of Stabilizer Circuits”. In: *Physical Review A* 70.5 (Nov. 30, 2004), p. 052328. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.70.052328. arXiv: quant-ph/0406196.

[31] David Winderl et al. *Architecture-Aware Synthesis of Stabilizer Circuits from Clifford Tableaus*. Oct. 30, 2024. DOI: 10.48550/arXiv.2309.08972. arXiv: 2309.08972[quant-ph].

[32] Sergey Bravyi et al. “Clifford Circuit Optimization with Templates and Symbolic Pauli Gates”. In: *Quantum* 5 (Nov. 16, 2021), p. 580. ISSN: 2521-327X. DOI: 10.22331/q-2021-11-16-580. arXiv: 2105.02291[quant-ph].

[33] Vadym Kliuchnikov and Dmitri Maslov. “Optimization of Clifford Circuits”. In: *Physical Review A* 88.5 (Nov. 8, 2013), p. 052307. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.88.052307. arXiv: 1305.0810[quant-ph].

[34] Hector J. Garcia, Igor L. Markov, and Andrew W. Cross. *Efficient Inner-product Algorithm for Stabilizer States*. Aug. 7, 2013. DOI: 10.48550/arXiv.1210.6646. arXiv: 1210.6646[cs].

[35] Stefan Krastanov et al. *QuantumSavory/QuantumClifford.jl: v0.11.5*. Version v0.11.5. June 2026. DOI: 10.5281/zenodo.20674456. URL: https://doi.org/10.5281/zenodo.20674456.

[36] Craig Gidney. “Stim: a fast stabilizer circuit simulator”. In: *Quantum* 5 (July 2021), p. 497. ISSN: 2521-327X. DOI: 10.22331/q-2021-07-06-497. URL: https://doi.org/10.22331/q-2021-07-06-497.

[37] Sergey Bravyi and Dmitri Maslov. “Hadamard-Free Circuits Expose the Structure of the Clifford Group”. In: *IEEE Transactions on Information Theory* 67.7 (2021), pp. 4546–4563. DOI: 10.1109/TIT.2021.3081415.

[38] Narayanan Rengaswamy et al. “Logical Clifford Synthesis for Stabilizer Codes”. In: *IEEE Transactions on Quantum Engineering* 1 (2020), pp. 1–17. ISSN: 2689-1808. DOI: 10.1109/tqe.2020.3023419. URL: http://dx.doi.org/10.1109/TQE.2020.3023419.

[39] Collin Beaudoin, Koustubh Phalak, and Swaroop Ghosh. *AltGraph: Redesigning Quantum Circuits Using Generative Graph Models for Efficient Optimization*. Mar. 21, 2024. DOI: 10.48550/arXiv.2403.12979. arXiv: 2403.12979[quant-ph].

[40] Qing-Hong Cao et al. *Quantum State Preparation via Large-Language-Model-Driven Evolution*. May 9, 2025. DOI: 10.48550/arXiv.2505.06347. arXiv: 2505.06347[quant-ph].

[41] A.Yu. Kitaev. “Fault-tolerant quantum computation by anyons”. In: *Annals of Physics* 303.1 (Jan. 2003), pp. 2–30. ISSN: 0003-4916. DOI: 10.1016/s0003-4916(02)00018-0. URL: http://dx.doi.org/10.1016/S0003-4916(02)00018-0.

[42] Nikolas P. Breuckmann and Jens Niklas Eberhardt. “Quantum Low-Density Parity-Check Codes”. In: *PRX Quantum* 2.4 (Oct. 2021). ISSN: 2691-3399. DOI: 10.1103/prxquantum.2.040101. URL: http://dx.doi.org/10.1103/PRXQuantum.2.040101.

[43] Andrey Boris Khesin and Jonathan Z. Lu. *Mirror codes: High-threshold quantum LDPC codes beyond the CSS regime*. 2026. arXiv: 2603.05496 [quant-ph]. URL: https://arxiv.org/abs/2603.05496.

[44] Laura Pecorari et al. “High-rate quantum LDPC codes for long-range-connected neutral atom registers”. In: *Nature Communications* 16.1 (Jan. 2025). ISSN: 2041–1723. DOI: 10.1038/s41467-025-56255-5. URL: http://dx.doi.org/10.1038/s41467-025-56255-5.

[45] John Stack, Ming Wang, and Frank Mueller. *Transversal Fault Tolerant Distributed Quantum Computing Operations*. 2026. arXiv: 2504.05611 [quant-ph]. URL: https://arxiv.org/abs/2504.05611.

[46] Robert G Gallager. *Low density parity check codes*. 1960. URL: http://hdl.handle.net/1721.1/11804.

[47] Qian Xu et al. *Batched high-rate logical operations for quantum LDPC codes*. 2025. arXiv: 2510.06159 [quant-ph]. URL: https://arxiv.org/abs/2510.06159.

[48] Theodore J. Yoder et al. *Tour de gross: A modular quantum computer based on bivariate bicycle codes*. June 3, 2025. DOI: 10.48550/arXiv.2506.03094. arXiv: 2506.03094[quant-ph].

[49] Naoyuki Kanomata and Hayato Goto. *Fault-tolerant quantum computing with a high-rate symplectic double code*. 2025. arXiv: 2509.15457 [quant-ph]. URL: https://arxiv.org/abs/2509.15457.

[50] Feroz Ahmed Mian, Owen Gwilliam, and Stefan Krastanov. *Multivariate Multi-cycle Codes for Complete Single-Shot Decoding*. 2026. arXiv: 2601.18879 [quant-ph]. URL: https://arxiv.org/abs/2601.18879.

[51] Boren Gu et al. *QGPU: Parallel logic in quantum LDPC codes*. 2026. arXiv: 2603.05398 [quant-ph]. URL: https://arxiv.org/abs/2603.05398.

[52] Lukas Voss et al. *Multivariate Bicycle Codes*. 2025. arXiv: 2406.19151 [quant-ph]. URL: https://arxiv.org/abs/2406.19151.

[53] Christian Kraglund Andersen and Eliška Greplová. *Small Quantum Low Density Parity Check Codes for Near-Term Experiments*. 2026. arXiv: 2507.09690 [quant-ph]. URL: https://arxiv.org/abs/2507.09690.

[54] Min Ye and Nicolas Delfosse. “Quantum error correction for long chains of trapped ions”. In: *Quantum* 9 (Nov. 2025), p. 1920. ISSN: 2521-327X. DOI: 10.22331/q-2025-11-27-1920. URL: http://dx.doi.org/10.22331/q-2025-11-27-1920.

[55] Ming Wang and Frank Mueller. “Coprime Bivariate Bicycle Codes and Their Layouts on Cold Atoms”. In: *Quantum* 10 (Feb. 2026), p. 2009. ISSN: 2521-327X. DOI: 10.22331/q-2026-02-23-2009. URL: http://dx.doi.org/10.22331/q-2026-02-23-2009.

[56] Narayanan Rengaswamy et al. “Synthesis of Logical Clifford Operators via Symplectic Geometry”. In: *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 791–795. DOI: 10.1109/isit.2018.8437652. URL: http://dx.doi.org/10.1109/ISIT.2018.8437652.

[57] Sergey Bravyi and Alexei Kitaev. “Universal quantum computation with ideal Clifford gates and noisy ancillas”. In: *Phys. Rev. A* 71 (2 Feb. 2005), p. 022316. DOI: 10.1103/PhysRevA.71.022316. URL: https://link.aps.org/doi/10.1103/PhysRevA.71.022316.

[58] Pedro Sales Rodriguez et al. “Experimental Demonstration of Logical Magic State Distillation”. In: *Nature* 645.8081 (Sept. 18, 2025), pp. 620–625. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-025-09367-3. arXiv: 2412.15165[quant-ph]. URL: http://arxiv.org/abs/2412.15165 (visited on 01/29/2026).

[59] Daniel Litinski. “A Game of Surface Codes: Large–Scale Quantum Computing with Lattice Surgery”. In: *Quantum* 3 (Mar. 5, 2019), p. 128. ISSN: 2521-327X. DOI: 10.22331/q-2019-03-05-128. arXiv: 1808.02892[quant-ph]. URL: http://arxiv.org/abs/1808.02892 (visited on 01/29/2026).

[60] Friederike Butt et al. “Measurement-free, scalable, and fault-tolerant universal quantum computing”. In: *Science Advances* 11.33 (2025), eadv2590. DOI: 10.1126/sciadv.adv2590. URL: https://www.science.org/doi/abs/10.1126/sciadv.adv2590.

[61] John Preskill. “Reliable Quantum Computers”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454.1969 (Jan. 8, 1998), pp. 385–410. ISSN: 1364-5021, 1471-2946. DOI: 10.1098/rspa.1998.0167. arXiv: quant-ph/9705031. URL: http://arxiv.org/abs/quant-ph/9705031 (visited on 02/05/2026).

[62] Daniel Gottesman. *An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation*. Apr. 16, 2009. DOI: 10.48550/arXiv.0904.2557. arXiv: 0904.2557[quant-ph].

[63] Hayato Goto. “Minimizing resource overheads for fault-tolerant preparation of encoded states of the Steane code”. In: *Scientific Reports* 6.1 (Jan. 27, 2016), p. 19578. ISSN: 2045-2322. DOI: 10.1038/srep19578. URL: https://doi.org/10.1038/srep19578.

[64] Rui Chao and Ben W. Reichardt. “Quantum error correction with only two extra qubits”. In: *Physical Review Letters* 121.5 (Aug. 1, 2018), p. 050502. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.121.050502. arXiv: 1705.02329[quant-ph]. URL: http://arxiv.org/abs/1705.02329 (visited on 02/05/2026).

[65] Alastair Kay. *Tutorial on the Quantikz Package*. 2026. arXiv: 1809.03842 [quant-ph]. URL: https://arxiv.org/abs/1809.03842.

[66] Peter-Jan H.S. Derks et al. “Designing fault-tolerant circuits using detector error models”. In: *Quantum* 9 (Nov. 2025), p. 1905. ISSN: 2521-327X. DOI: 10.22331/q-2025-11-06-1905. URL: http://dx.doi.org/10.22331/q-2025-11-06-1905.

[67] Diego Forlivesi and David Amaro. *Flag at origin: a modular fault-tolerant preparation for CSS codes*. 2026. arXiv: 2508.14200 [quant-ph]. URL: https://arxiv.org/abs/2508.14200.

[68] Rajeev Acharya et al. “Quantum error correction below the surface code threshold”. In: *Nature* 638.8052 (Feb. 1, 2025), pp. 920–926. ISSN: 1476-4687. DOI: 10.1038/s41586-024-08449-y. URL: https://doi.org/10.1038/s41586-024-08449-y.

[69] P.W. Shor. “Fault-tolerant quantum computation”. In: *Proceedings of 37th Conference on Foundations of Computer Science*. 1996, pp. 56–65. DOI: 10.1109/SFCS.1996.548464.

[70] D. Aharonov and M. Ben-Or. “Fault–tolerant quantum computation with constant error”. In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. STOC ’97. El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 176–188. ISBN: 0897918886. DOI: 10.1145/258533.258579. URL: https://doi.org/10.1145/258533.258579.

[71] Emanuel Knill, Raymond Laflamme, and Wojciech H. Zurek. “Resilient quantum computation: error models and thresholds”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454.1969 (Jan. 1998), pp. 365–384. ISSN: 1471-2946. DOI: 10.1098/rspa.1998.0166. URL: http://dx.doi.org/10.1098/rspa.1998.0166.

[72] Hasan Sayginel et al. “Fault-Tolerant Logical Clifford Gates from Code Automorphisms”. In: *PRX Quantum* 6.3 (Sept. 5, 2025), p. 030343. ISSN: 2691-3399. DOI: 10.1103/vf7v-cpq9. URL: https://link.aps.org/doi/10.1103/vf7v-cpq9 (visited on 01/28/2026).

[73] Zijian Liang and Yu-An Chen. *Self-dual bivariate bicycle codes with transversal Clifford gates*. Jan. 11, 2026. DOI: 10.48550/arXiv.2510.05211. arXiv: 2510.05211[quant-ph]. URL: http://arxiv.org/abs/2510.05211 (visited on 01/30/2026).

[74] Bryan Eastin and Emanuel Knill. *Restrictions on Transversal Encoded Quantum Gate Sets*. July 9, 2009. DOI: 10.1103/PhysRevLett.102.110502. arXiv: 0811.4262[quant-ph]. URL: http://arxiv.org/abs/0811.4262 (visited on 01/30/2026).

[75] Earl T. Campbell, Barbara M. Terhal, and Christophe Vuillot. “Roads towards fault-tolerant universal quantum computation”. In: *Nature* 549.7671 (2017), pp. 172–179. ISSN: 1476-4687. DOI: 10.1038/nature23460. URL: http://dx.doi.org/10.1038/nature23460.

[76] Paul Webster et al. “Universal fault-tolerant quantum computing with stabilizer codes”. In: *Phys. Rev. Res.* 4 (1 Feb. 2022), p. 013092. DOI: 10.1103/PhysRevResearch.4.013092.

[77] Siddhant Singh et al. “Modular Architectures and Entanglement Schemes for Error-Corrected Distributed Quantum Computation”. In: *npj Quantum Information* 12.1 (Dec. 2, 2025), p. 3. ISSN: 2056-6387. DOI: 10.1038/s41534-025-01146-2. arXiv: 2408.02837[quant-ph].

[78] Richard Cleve and Daniel Gottesman. “Efficient computations of encodings for quantum error correction”. In: *Physical Review A* 56.1 (1997), pp. 76–82. ISSN: 1094-1622. DOI: 10.1103/physreva.56.76. URL: http://dx.doi.org/10.1103/PhysRevA.56.76.

[79] Andrew M. Steane. *Fast fault-tolerant filtering of quantum codewords*. 2004. arXiv: quant-ph/0202036 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0202036.

[80] Adam Paetznick and Ben W. Reichardt. *Fault-tolerant ancilla preparation and noise threshold lower bounds for the 23-qubit Golay code*. Apr. 12, 2013. DOI: 10.48550/arXiv.1106.2190. arXiv: 1106.2190[quant-ph]. URL: http://arxiv.org/abs/1106.2190 (visited on 02/05/2026).

[81] Arijit Mondal and Keshab K. Parhi. “Quantum Circuits for Stabilizer Error Correcting Codes: A Tutorial”. In: *IEEE Circuits and Systems Magazine* 24.1 (2024), pp. 33–51. ISSN: 1531-636X, 1558-0830. DOI: 10.1109/MCAS.2024.3349668. arXiv: 2309.11793[quant-ph]. URL: http://arxiv.org/abs/2309.11793 (visited on 04/28/2026).

[82] Jan Olle et al. “Simultaneous discovery of quantum error correction codes and encoders with a noise-aware reinforcement learning agent”. In: *npj Quantum Information* 10.1 (Dec. 2024). ISSN: 2056-6387. DOI: 10.1038/s41534-024-00920-y. URL: http://dx.doi.org/10.1038/s41534-024-00920-y.

[83] Christopher Chamberland and Michael E. Beverland. “Flag fault-tolerant error correction with arbitrary distance codes”. In: *Quantum* 2 (2018), p. 53. ISSN: 2521-327X. DOI: 10.22331/q-2018-02-08-53. URL: http://dx.doi.org/10.22331/q-2018-02-08-53.

[84] Rui Chao and Ben W. Reichardt. “Flag fault-tolerant error correction for any stabilizer code”. In: *PRX Quantum* 1.1 (Sept. 3, 2020), p. 010302. ISSN: 2691-3399. DOI: 10.1103/PRXQuantum.1.010302. arXiv: 1912.09549[quant-ph].

[85] Dhruv Bhatnagar et al. “Low-Depth Flag-Style Syndrome Extraction for Small Quantum Error-Correction Codes”. In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, Sept. 2023, pp. 63–69. DOI: 10.1109/qce57702.2023.00016. URL: http://dx.doi.org/10.1109/QCE57702.2023.00016.

[86] Sascha Heußen, David F. Locher, and Markus Müller. “Measurement-Free Fault-Tolerant Quantum Error Correction in Near-Term Devices”. In: *PRX Quantum* 5.1 (Feb. 2024). ISSN: 2691-3399. DOI: 10.1103/prxquantum.5.010333. URL: http://dx.doi.org/10.1103/PRXQuantum.5.010333.

[87] Hayato Goto, Yinghao Ho, and Taro Kanao. “Measurement-free fault-tolerant logical zero-state encoding of the distance-three nine-qubit surface code in a one-dimensional qubit array”. In: *Physical Review Research* 5.4 (Nov. 9, 2023), p. 043137. ISSN: 2643-1564. DOI: 10.1103/PhysRevResearch.5.043137. arXiv: 2303.17211[quant-ph]. URL: http://arxiv.org/abs/2303.17211 (visited on 02/02/2026).

[88] C. Ryan-Anderson et al. “Realization of Real-Time Fault-Tolerant Quantum Error Correction”. In: *Phys. Rev. X* 11 (4 Dec. 2021), p. 041058. DOI: 10.1103/PhysRevX.11.041058. URL: https://link.aps.org/doi/10.1103/PhysRevX.11.041058.

[89] Lucas Daguerre et al. “Experimental Demonstration of High-Fidelity Logical Magic States from Code Switching”. In: *Physical Review X* 15.4 (Oct. 2025). ISSN: 2160-3308. DOI: 10.1103/dck4-x9c2. URL: http://dx.doi.org/10.1103/dck4-x9c2.

[90] Robert Beals et al. “Efficient distributed quantum computing”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 469.2153 (May 2013). ISSN: 1471-2946. DOI: 10.1098/rspa.2012.0686. URL: http://dx.doi.org/10.1098/rspa.2012.0686.

[91] Jay Gambetta. *Quantum-centric supercomputing: The next wave of computing*. IBM Quantum Research Blog. Nov. 2022. URL: https://www.ibm.com/quantum/blog/next-wave-quantum-centric-supercomputing (visited on 08/17/2026).

[92] Eduard Alarcón et al. “Scalable multi-chip quantum architectures enabled by cryogenic hybrid wireless/quantum-coherent network-in-package”. In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2023, pp. 1–5. DOI: 10.1109/ISCAS46773.2023.10181857.

[93] IBM. *The future of computing is quantum-centric*. URL: https://www.ibm.com/roadmaps/quantum/ (visited on 06/23/2026).

[94] IonQ. *Industry-Leading Roadmap*. URL: https://www.ionq.com/roadmap (visited on 06/23/2026).

[95] Government of Canada. *National Quantum Strategy roadmap: Quantum computing*. 2025. URL: https://ised-isde.canada.ca/site/national-quantum-strategy/en/national-quantum-strategy-roadmap-quantum-computing (visited on 06/23/2026).

[96] Hana KimLee et al. *QuantumSavory: Write Symbolically, Run on Any Backend – A Unified Simulation Toolkit for Quantum Computing and Networking*. Dec. 18, 2025. DOI: 10.48550/arXiv.2512.16752. arXiv: 2512.16752[quant-ph].

[97] Tim Coopmans et al. “NetSquid, a NETwork Simulator for QUantum Information using Discrete events”. In: *Communications Physics* 4.1 (July 2021). ISSN: 2399-3650. DOI: 10.1038/s42005-021-00647-8. URL: http://dx.doi.org/10.1038/s42005-021-00647-8.

[98] David Barral et al. “Review of Distributed Quantum Computing: From single QPU to High Performance Quantum Computing”. In: *Computer Science Review* 57 (Aug. 2025), p. 100747. ISSN: 15740137. DOI: 10.1016/j.cosrev.2025.100747.

[99] Davide Ferrari et al. “Compiler Design for Distributed Quantum Computing”. In: *IEEE Transactions on Quantum Engineering* 2 (2021), pp. 1–20. ISSN: 2689-1808. DOI: 10.1109/TQE.2021.3053921. arXiv: 2012.09680[quant-ph].

[100] D. Main et al. “Distributed quantum computing across an optical network link”. In: *Nature* 638.8050 (Feb. 13, 2025), pp. 383–388. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-024-08404-x.

[101] J. Eisert et al. “Optimal local implementation of nonlocal quantum gates”. In: *Physical Review A* 62.5 (Oct. 2000). ISSN: 1094-1622. DOI: 10.1103/physreva.62.052317. URL: http://dx.doi.org/10.1103/PhysRevA.62.052317.

[102] Pablo Andrés-Martínez and Chris Heunen. “Automated distribution of quantum circuits via hypergraph partitioning”. In: *Physical Review A* 100.3 (Sept. 5, 2019), p. 032308. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.100.032308. arXiv: 1811.10972[quant-ph].

[103] Enrico Russo et al. “TeleSABRE: Heuristic Layout Synthesis in Multi-Core Quantum Systems with Teleport Interconnect”. In: *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, Aug. 2025, pp. 749–758. DOI: 10.1109/qce65121.2025.00086.

[104] Jwo-Sy Chen et al. “Benchmarking a trapped-ion quantum computer with 30 qubits”. In: *Quantum* 8 (Nov. 2024), p. 1516. ISSN: 2521-327X. DOI: 10.22331/q-2024-11-07-1516. URL: http://dx.doi.org/10.22331/q-2024-11-07-1516.

[105] Naomi H. Nickerson, Ying Li, and Simon C. Benjaming. “Topological quantum computing with a very noisy network and local error rates approaching one percent”. In: *Nature Communications* 4 (2013). DOI: 10.1038/ncomms2773.

[106] Sebastian Brandhofer, Ilia Polian, and Kevin Krsulich. “Optimal Partitioning of Quantum Circuits Using Gate Cuts and Wire Cuts”. In: *IEEE Transactions on Quantum Engineering* 5 (2024), pp. 1–10. DOI: 10.1109/TQE.2023.3347106.

[107] Agata M. Brańczyk et al. *Qiskit addon: circuit cutting*. https://github.com/Qiskit/qiskit-addon-cutting. 2024. DOI: 10.5281/zenodo.7987997.

[108] Jobit Varughese. *What is LLM training?* IBM. URL: https://www.ibm.com/think/topics/llm-training (visited on 06/29/2026).

[109] Pablo Andres-Martinez et al. “Distributing circuits over heterogeneous, modular quantum computing network architectures”. In: *Quantum Science and Technology* 9.4 (Oct. 1, 2024), p. 045021. ISSN: 2058-9565. DOI: 10.1088/2058-9565/ad6734.

[110] Jonathan M. Baker et al. “Time-Sliced Quantum Circuit Partitioning for Modular Architectures”. In: *Proceedings of the 17th ACM International Conference on Computing Frontiers*. May 11, 2020, pp. 98–107. DOI: 10.1145/3387902.3392617. arXiv: 2005.12259[quant-ph].

[111] Medina Bandic et al. “Mapping quantum circuits to modular architectures with QUBO”. In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Sept. 17, 2023, pp. 790–801. DOI: 10.1109/QCE57702.2023.00094. arXiv: 2305.06687[quant-ph].

[112] Pau Escofet et al. “Revisiting the Mapping of Quantum Circuits: Entering the Multi-Core Era”. In: *ACM Transactions on Quantum Computing* 6.1 (Mar. 31, 2025), pp. 1–26. ISSN: 2643-6809, 2643-6817. DOI: 10.1145/3655029. arXiv: 2403.17205[quant-ph].

[113] Felix Burt, Kuan-Cheng Chen, and Kin K. Leung. “Generalised Circuit Partitioning for Distributed Quantum Computing”. In: *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2024, pp. 173–178. DOI: 10.1109/qce60285.2024.10273. URL: http://dx.doi.org/10.1109/QCE60285.2024.10273.

[114] Felix Burt, Kuan-Cheng Chen, and Kin K. Leung. “A Multilevel Framework for Partitioning Quantum Circuits”. In: *Quantum* 10 (Jan. 2026), p. 1984. ISSN: 2521-327X. DOI: 10.22331/q-2026-01-22-1984. URL: http://dx.doi.org/10.22331/q-2026-01-22-1984.

[115] Enrico Russo et al. *Attention-Based Deep Reinforcement Learning for Qubit Allocation in Modular Quantum Architectures*. June 17, 2024. DOI: 10.48550/arXiv.2406.11452. arXiv: 2406.11452[quant-ph].

[116] Panagiotis Promponas et al. *Compiler for Distributed Quantum Computing: a Reinforcement Learning Approach*. Apr. 25, 2024. DOI: 10.48550/arXiv.2404.17077. arXiv: 2404.17077[quant-ph].

[117] Sokea Sang, Leanghok Hour, and Youngsun Han. *Toward Scalable Quantum Compilation for Modular Architecture: Qubit Mapping and Reuse via Deep Reinforcement Learning*. Aug. 13, 2025. DOI: 10.48550/arXiv.2506.09323. arXiv: 2506.09323[quant-ph].

[118] Leo Sünkel, Manik Dawar, and Thomas Gabor. *Applying an Evolutionary Algorithm to Minimize Teleportation Costs in Distributed Quantum Computing*. Sept. 17, 2024. DOI: 10.48550/arXiv.2311.18529. arXiv: 2311.18529[quant-ph].

[119] Haozhen Situ et al. “Distributed quantum architecture search”. In: *Physical Review A* 110.2 (Aug. 1, 2024), p. 022403. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.110.022403. arXiv: 2403.06214[quant-ph].

[120] Leo Sünkel et al. *Time-Aware Qubit Assignment and Circuit Optimization for Distributed Quantum Computing*. July 15, 2025. DOI: 10.48550/arXiv.2507.11707. arXiv: 2507.11707[quant-ph].

[121] Cuomo. “Optimized Compiler for Distributed Quantum Computing”. In: *ACM Transactions on Quantum Computing* 4.2 (Feb. 2023), Article 15. DOI: https://doi.org/10.1145/3579367.

[122] Anbang Wu, Yufei Ding, and Ang Li. “QuComm: Optimizing Collective Communication for Distributed Quantum Computing”. In: *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23)* (2023). DOI: https://doi.org/10.1145/3613424.3614253.

[123] Ranjani G. Sundaram and Himanshu Gupta. “Dynamic Distribution of Quantum Circuits with Minimum Circuit-Execution Time”. In: *2025 International Conference on Quantum Communications, Networking, and Computing (QCNC)*. 2025, pp. 238–246.

[124] Johannes Knörzer et al. *Distributed Quantum Information Processing: A Review of Recent Progress*. Oct. 17, 2025. DOI: 10.48550/arXiv.2510.15630. arXiv: 2510.15630[quant-ph].

[125] Evan Sutcliffe et al. "Distributed Quantum Error Correction Based on Hyperbolic Floquet Codes". In: *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, Aug. 2025, pp. 649–657. DOI: 10.1109/qce65121.2025.00076. URL: http://dx.doi.org/10.1109/QCE65121.2025.00076.

[126] M. Canteri et al. "Photon-Interfaced Ten-Qubit Register of Trapped Ions". In: *Phys. Rev. Lett.* 135 (8 Aug. 2025), p. 080801. DOI: 10.1103/v5k1-whwz. URL: https://link.aps.org/doi/10.1103/v5k1-whwz.

[127] Nitish Kumar Chandra, Eneet Kaur, and Kaushik P. Seshadreesan. "Architectural Approaches to Fault-Tolerant Distributed Quantum Computing and Their Entanglement Overheads". In: *2025 IEEE 7th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*. IEEE, Nov. 2025, pp. 561–572. DOI: 10.1109/tps-isa67132.2025.00068. URL: http://dx.doi.org/10.1109/TPS-ISA67132.2025.00068.

[128] Sébastian de Bone et al. "Thresholds for the distributed surface code in the presence of memory decoherence". In: *AVS Quantum Science* 6.3 (July 2024). ISSN: 2639-0213. DOI: 10.1116/5.0200190. URL: http://dx.doi.org/10.1116/5.0200190.

[129] Nitish Kumar Chandra et al. "Distributed Realization of Color Codes for Quantum Error Correction". In: *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, Aug. 2025, pp. 2482–2492. DOI: 10.1109/qce65121.2025.00269. URL: http://dx.doi.org/10.1109/QCE65121.2025.00269.

[130] Edwin Tham et al. *Distributed fault-tolerant quantum memories over a 2xL array of qubit modules*. 2025. arXiv: 2508.01879 [quant-ph]. URL: https://arxiv.org/abs/2508.01879.

[131] Joshua Ramette et al. *Fault-Tolerant Connection of Error-Corrected Qubits with Noisy Links*. Feb. 2, 2023. DOI: 10.48550/arXiv.2302.01296. arXiv: 2302.01296[quant-ph].

[132] Markus Grassl. "Variations on Encoding Circuits for Stabilizer Quantum Codes". In: *Coding and Cryptology*. Ed. by Yeow Meng Chee et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 142–158. ISBN: 978-3-642-20901-7.

[133] Dongxiao Quan et al. "Implementation of Fault-Tolerant Encoding Circuit Based on Stabilizer Implementation and 'Flag' Bits in Steane Code". In: *Entropy* 24.1107 (2022). ISSN: 1099-4300. DOI: 10.3390/e24081107. URL: https://www.mdpi.com/1099-4300/24/8/1107.

[134] Sokea Sang, Leanghok Hour, and Youngsun Han. "Learning-optimized qubit mapping and reuse to minimize intercore communication in modular quantum architectures". In: *Physical Review Applied* 25.5 (May 2026). ISSN: 2331-7019. DOI: 10.1103/rfn7-c5n6. URL: http://dx.doi.org/10.1103/rfn7-c5n6.

[135] Evan Sutcliffe and Coral M. Westoby. *Tolerating Device Failure in Distributed Quantum Computing*. 2026. arXiv: 2605.11088 [quant-ph]. URL: https://arxiv.org/abs/2605.11088.

[136] Shin Ho Choe et al. *Barbell Codes: qLDPC Codes for Superconducting Quantum Hardware*. 2026. arXiv: 2606.06062 [quant-ph]. URL: https://arxiv.org/abs/2606.06062.

[137] Darya Martyniuk, Johannes Jung, and Adrian Paschke. "Quantum Architecture Search: A Survey". In: *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2024, pp. 1695–1706. DOI: 10.1109/qce60285.2024.00198. URL: http://dx.doi.org/10.1109/QCE60285.2024.00198.

[138] Ketan N. Patel, Igor L. Markov, and John P. Hayes. "Optimal synthesis of linear reversible circuits". In: *Quantum Info. Comput.* 8.3 (Mar. 2008), pp. 282–294. ISSN: 1533-7146.

[139] Gongyu Ni et al. *Advanced Scheduling Strategies for Distributed Quantum Computing Jobs*. 2026. arXiv: 2602.24152 [quant-ph]. URL: https://arxiv.org/abs/2602.24152.

[140] Luise Prielinger et al. "Piecemaker: a resource-efficient entanglement distribution protocol". In: *IEEE Journal on Selected Areas in Communications* (2026), pp. 1–1. ISSN: 0733-8716. DOI: 10.1109/jsac.2026.3704337. URL: http://dx.doi.org/10.1109/JSAC.2026.3704337.

[141] Dmitri Maslov. "Basic circuit compilation techniques for an ion-trap quantum machine". In: *New Journal of Physics* 19.2 (Feb. 2017), p. 023035. ISSN: 1367-2630. DOI: 10.1088/1367-2630/aa5e47. URL: http://dx.doi.org/10.1088/1367-2630/aa5e47.

[142] Sebastian Schlag et al. "High-Quality Hypergraph Partitioning". In: *ACM J. Exp. Algorithmics* (Mar. 2022). DOI: 10.1145/3529090. URL: https://doi.org/10.1145/3529090.

[143] Jun-Yi Wu et al. "Entanglement-efficient bipartite-distributed quantum computing". In: *Quantum* 7 (Dec. 5, 2023), p. 1196. ISSN: 2521-327X. DOI: 10.22331/q-2023-12-05-1196. arXiv: 2212.12688[quant-ph].

[144] John H. Holland. "Genetic Algorithms". In: *Scientific American* 267.1 (1992), pp. 66–73. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24939139 (visited on 07/07/2026).

[145] Mark A. Webster and Dan E. Browne. "Engineering Quantum Error Correction Codes Using Evolutionary Algorithms". In: *IEEE Transactions on Quantum Engineering* 6 (2025), pp. 1–14. DOI: 10.1109/TQE.2025.3538934.

[146] Daniel Tandeitnik and Thiago Guerreiro. "Evolving quantum circuits". In: *Quantum Information Processing* 23.3 (Mar. 2024). ISSN: 1573-1332. DOI: 10.1007/s11128-024-04317-w. URL: http://dx.doi.org/10.1007/s11128-024-04317-w.

[147] Sünkel, Leo and Stein, Jonas and Stenzel, Gerhard and Kölle, Michael and Gabor, Thomas and Linnhoff-Popien, Claudia. *Evolutionary-Based Circuit Optimization for Distributed Quantum Computing*. Sept. 9, 2025. DOI: 10.48550/arXiv.2509.08074. arXiv: 2509.08074[quant-ph].

[148] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (Oct. 1, 2017), pp. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270.

[149] Peiyong Wang et al. “Automated Quantum Circuit Design With Nested Monte Carlo Tree Search”. In: *IEEE Transactions on Quantum Engineering* 4 (2023), pp. 1–20. ISSN: 2689-1808. DOI: 10.1109/tqe.2023.3265709. URL: http://dx.doi.org/10.1109/TQE.2023.3265709.

[150] Maxim Egorov et al. “POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty”. In: *Journal of Machine Learning Research* 18.26 (2017), pp. 1–5. URL: http://jmlr.org/papers/v18/16-300.html.

[151] Ali Javadi-Abhari et al. *Quantum computing with Qiskit*. 2024. DOI: 10.48550/arXiv.2405.08810. arXiv: 2405.08810 [quant-ph].

[152] Robert Wille et al. “The MQT Handbook : A Summary of Design Automation Tools and Software for Quantum Computing”. In: *2024 IEEE International Conference on Quantum Software (QSW)*. IEEE, 2024, pp. 1–8. DOI: 10.1109/qsw62656.2024.00013. URL: http://dx.doi.org/10.1109/QSW62656.2024.00013.

[153] Michele Bandini et al. “Optimized Compilation for Distributed Quantum Computing”. In: *IEEE Access* 14 (Jan. 2026), pp. 97220–97231. DOI: 10.1109/ACCESS.2026.3706203.

[154] Andrew J. Landahl, Jonas T. Anderson, and Patrick R. Rice. *Fault-tolerant quantum computing with color codes*. 2011. arXiv: 1108.5738 [quant-ph]. URL: https://arxiv.org/abs/1108.5738.

[155] Vincenzo Lipardi et al. “Quantum Circuit Design using a Progressive Widening Enhanced Monte Carlo Tree Search”. In: *Advanced Quantum Technologies* 8.10 (2025). ISSN: 2511-9044. DOI: 10.1002/qute.202500093. URL: http://dx.doi.org/10.1002/qute.202500093.

[156] Tom Peham et al. *Synthesis and Optimization of Encoding Circuits for Fault-Tolerant Quantum Computation*. 2026. arXiv: 2605.15266 [quant-ph]. URL: https://arxiv.org/abs/2605.15266.

[157] Rui Chao and Ben W. Reichardt. “Quantum Error Correction with Only Two Extra Qubits”. In: *Physical Review Letters* 121.5 (Aug. 2018). ISSN: 1079-7114. DOI: 10.1103/physrevlett.121.050502. URL: http://dx.doi.org/10.1103/PhysRevLett.121.050502.

[158] Konstantinos Georgopoulos, Clive Emary, and Paolo Zuliani. “Modeling and simulating the noisy behavior of near-term quantum computers”. In: *Physical Review A* 104.6 (Dec. 2021). ISSN: 2469-9934. DOI: 10.1103/physreva.104.062432. URL: http://dx.doi.org/10.1103/PhysRevA.104.062432.

[159] Quantinuum. *Quantinuum Helios Product Data Sheet*. Version 1.2. June 2026. URL: https://docs.quantinuum.com/systems/_static/assets/data_sheets/Quantinuum%20Helios%20Product%20Data%20Sheet.pdf (visited on 07/24/2026).

[160] Quantinuum. *Quantinuum Systems Performance Validation – Component Benchmarks*. 2025. URL: https://docs.quantinuum.com/systems/user_guide/hardware_user_guide/performance_validation.html (visited on 07/24/2026).

[161] L.J. Stephenson et al. “High-Rate, High-Fidelity Entanglement of Qubits Across an Elementary Quantum Network”. In: *Physical Review Letters* 124.11 (Mar. 2020). ISSN: 1079-7114. DOI: 10.1103/physrevlett.124.110501. URL: http://dx.doi.org/10.1103/PhysRevLett.124.110501.

[162] Sagnik Saha et al. “High-fidelity remote entanglement of trapped atoms mediated by time-bin photons”. In: *Nature Communications* 16.1 (Mar. 2025). ISSN: 2041-1723. DOI: 10.1038/s41467-025-57557-4. URL: http://dx.doi.org/10.1038/s41467-025-57557-4.

[163] Jameson O’Reilly et al. “Fast Photon-Mediated Entanglement of Continuously Cooled Trapped Ions for Quantum Networking”. In: *Physical Review Letters* 133.9 (Aug. 2024). ISSN: 1079-7114. DOI: 10.1103/physrevlett.133.090802. URL: http://dx.doi.org/10.1103/PhysRevLett.133.090802.

[164] IonQ. *IonQ Aria: Practical Performance*. Jan. 2025. URL: https://www.ionq.com/resources/ionq-aria-practical-performance (visited on 07/25/2026).

[165] Erik Weilandt, Tom Peham, and Robert Wille. *Synthesis of Fault-tolerant State Preparation Circuits using Steane-type Error Detection*. 2026. arXiv: 2601.13313 [quant-ph]. URL: https://arxiv.org/abs/2601.13313.

[166] Daniel Gottesman. *Errata in “Stabilizer Codes and Quantum Error Correction"*. 2004. URL: https://perimeterinstitute.ca/personal/dgottesman/thesis-errata.html (visited on 06/20/2026).

[167] Tim Neumann. *DQCode*. 2026. URL: https://github.com/timneumann1/DQCode/tree/main.

[168] Tim Miller. *Mastering Reinforcement Learning*. The University of Queensland, 2023. URL: https://gibberblot.github.io/rl-notes/index.html (visited on 07/12/2026).

[169] Lodewijk Kallenberg. *Markov Decision Processes*. Lecture Notes, University of Leiden. Oct. 2016. URL: https://pub.math.leidenuniv.nl/~kallenberglcm/Lecture-notes-MDP.pdf (visited on 08/18/2026).

[170] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (Feb. 1, 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://doi.org/10.1038/nature14236.

[171] Nikhil Sharma et al. *Introduction to Artificial Intelligence*. Textbook, CS 188: Introduction to Artificial Intelligence, UC Berkeley. Oct. 2024. URL: https://inst.eecs.berkeley.edu/~cs188/textbook/ (visited on 08/18/2026).

[172] Rémi Coulom. “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search”. In: *Computers and Games*. Ed. by H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 72–83. ISBN: 978-3-540-75538-8.

[173] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *Machine Learning: ECML 2006*. Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293. ISBN: 978-3-540-46056-5.

A DQCode Software

A.1 Data and Code Availability

All code used in this project is available in the open-source repository DQCode at https://github.com/timneumann1/DQCode/tree/main, and all data is accessible at https://github.com/timneumann1/DQCode_Data. The collection of code and data may additionally be accessed in the 4TU.ResearchData repository at https://doi.org/10.4121/8543d5ea-2ae8-4af2-84dd-4364799e9b7a. The structure of the repository is visualised in Appendix A.3.

A.2 Reproducibility

All optimisation and simulation results in this work were obtained with a random seed for reproducibility, and can be replicated by following the instructions provided in the DQCode codebase. All optimisation and simulation experiments were performed on a MacBook Air with a M3 chip (8-core).

A.3 DQCode Code Structure

The overall code structure of DQCode is summarised in Figure 36, which depicts dependencies of files.

Figure 36: Repository structure and file relations of `DQCode`. The `src/types.jl` and `src/helper.jl` files are referenced by all files in `src/`, with arrows being omitted in order to reduce visual clutter.

B Supplementary Information and Derivations

B.1 Stabiliser code projector

We can verify that the projector Π_C acts correctly on all quantum states $|\psi\rangle$ in $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$ by computing

$$\Pi_{S_i}|\psi\rangle = \frac{1}{2}(\mathbb{1}_{2^n} + S_i)|\psi\rangle = \frac{1}{2}(|\psi\rangle + |\psi\rangle) = |\psi\rangle \tag{63}$$

and

$$\Pi_{S_i}|\psi^\perp\rangle = \frac{1}{2}(\mathbb{1}_{2^n} + S_i)|\psi^\perp\rangle = \frac{1}{2}(|\psi^\perp\rangle - |\psi^\perp\rangle) = 0, \tag{64}$$

where $|\psi\rangle$ is any state in the $(+1)$-eigenspace of S_i, and $|\psi^\perp\rangle$ is a (-1)-eigenstate of S_i and thus lives in an orthogonal eigenspace of $\mathcal{S}$.

Now using a result from linear algebra, the dimension of the projection target space (C) equals the rank of the projection operator ($\Pi_\mathcal{S}$), which can be computed as the trace

$$\mathrm{Tr}(\Pi_\mathcal{S}) = \mathrm{Tr}\Big(\prod_{i=1}^{n-k}\frac{\mathbb{1}_{2^n}+S_i}{2}\Big) = \frac{1}{2^{n-k}}\mathrm{Tr}\Big[\prod_{i=1}^{n-k}\Big(\mathbb{1}_{2^n}+S_i\Big)\Big] = \frac{1}{2^{n-k}}\mathrm{Tr}\Big(\prod_{i=1}^{n-k}\mathbb{1}_{2^n}\Big) = 2^{n-(n-k)} = 2^k, \tag{65}$$

where we have used the fact that all Pauli operators (and all products thereof) have zero trace.

B.2 Stabiliser Measurement

In the main text, we saw that X- and Z-type measurements are crucial tools in QEC. This appendix provides the circuit implementation of both as special cases from Figure 2. In particular, we replace S with the respective stabiliser, and perform some circuit simplification for the Z-type measurement, based on the identity $HZH = X$. It should be noted that multi-qubit stabilisers *of the same type* can be measured easily by adding the respective controlled operations between the Hadamard sandwich in Circuits 37 and 38.

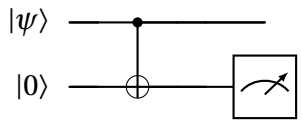


Figure 37: Z-type stabiliser measurement, detecting X-errors

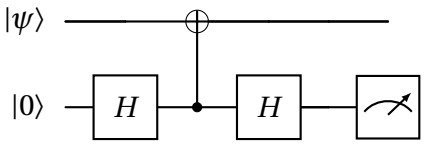


Figure 38: X-type stabiliser measurement, detecting Z-errors

B.3 Threshold theorem

The threshold theorem asserts that we can concatenate codes to achieve arbitrarily small errors rate at the cost of increased gate overhead. The below definition is taken from [3, Ch. 10].

Theorem 3: Threshold Theorem

A quantum circuit containing $p(n)$ gates may be simulated with probability of error at most ϵ using

$$O\left(\text{poly}\left[\log\frac{p(n)}{\epsilon}\right]p(n)\right)$$

gates on hardware whose components fail with probability at most p, provided p is below some constant threshold $p < p_{th}$, and given reasonable assumptions about the noise in the underlying hardware. (cf. [3, Ch. 10.6.1])

B.4 Transversal CX gate between two Steane codes

The below quantum circuit visualises a transversal CX gate between two Steane code modules.

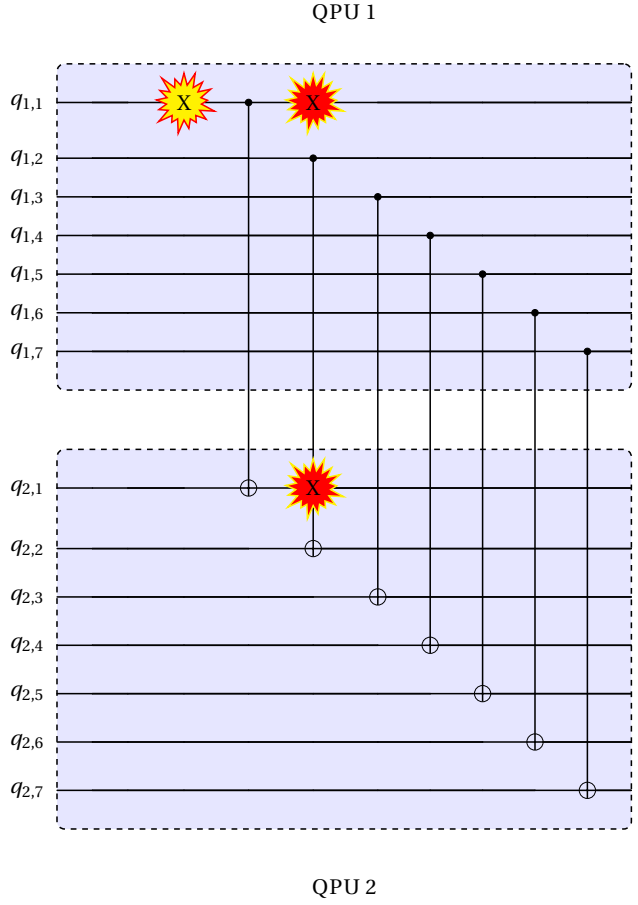


Figure 39: Fault-tolerant transversal CX gate between two Steane code encoding blocks of seven qubits each; a single X-error (red outline) on some qubit propagates to one X-error (yellow outline) in *each encoded block*.

B.5 Derivation of the Canonical Logical Operators

Gottesman introduced the standard form from Matrix 24 without reducing the submatrix right above the $\mathbb{1}_s$ block in his original work [17], as displayed below:

$$\mathrm{T} = \begin{matrix} \\ r \\ s \end{matrix} \left(\begin{array}{ccc|ccc} \overbrace{}^{r} & \overbrace{}^{s} & \overbrace{}^{k} & \overbrace{}^{r} & \overbrace{}^{s} & \overbrace{}^{k} \\ \mathbb{1}_r & A_1 & A_2 & B & C_1 & C_2 \\ 0 & 0 & 0 & D & \mathbb{1}_s & E \end{array} \right). \tag{66}$$

Here, A_1, A_2, B, C_1, C_2, D and E again denote the submatrices of indicated dimensions. In this derivation, we choose to follow Gottesman's original formulation, which later reduces to the expressions provided in the main text.

To determine the logical operators of a code corresponding to Tableau 66, let the logical X-operators be captured by the $k \times n$ matrix

$$[X_L] = \left(\begin{array}{ccc|ccc} \overbrace{}^{r} & \overbrace{}^{s} & \overbrace{}^{k} & \overbrace{}^{r} & \overbrace{}^{s} & \overbrace{}^{k} \\ U_1 & U_2 & U_3 & V_1 & V_2 & V_3 \end{array} \right). \tag{67}$$

We proceed stepwise to determine all six sub-matrices U_i and V_i, up to certain gauge freedoms exploited along the way. Since logical operators are defined modulo the stabiliser group, and the identity (all-zero row) is part of the stabiliser group, we may multiply by the appropriate stabilisers to bring the logical operators into the form

$$\left(\begin{array}{ccc|ccc} \overbrace{}^{r} & \overbrace{}^{s} & \overbrace{}^{k} & \overbrace{}^{r} & \overbrace{}^{s} & \overbrace{}^{k} \\ \mathbf{0}_{k\times r} & U_2 & U_3 & V_1 & \mathbf{0}_{k\times s} & V_3 \end{array} \right). \tag{68}$$

Next, we draw upon the requirement that all logical operators commute pairwise, and that any logical operator commutes with all stabilisers. For this, Gottesman provides an arithmetic condition for two operators to commute in the binary setting [17, Ch. 3.4]. Let $[\gamma|\delta]$ and $[\pi|\omega]$ be operators represented by their X and Z components of qubit size n, respectively. Then the two operators commute iff

$$\sum_{i=1}^{n} \gamma_i\omega_i + \delta_i\pi_i = 0 \tag{69}$$

holds, where addition is performed modulo 2. Condition 69 precisely captures that for two operators to commute, they need to share support on an even number of anti-commuting components. For example, the operators $X_1Z_2 = [10|01]$ and $I_1X_2 = [01|00]$ do not commute: The Paulis on qubit 1 commute, whereas they do not commute on qubit 2, yielding a sum of $(1\cdot 0 + 0\cdot 1) + (0\cdot 0 + 1\cdot 1) = 1$). On the other hand, the operators $X_1Z_2[10|01]$ and $Z_1X_2 = [01|10]$ do commute: For both qubits, the Paulis anti-commute (there is an even number of anti-commuting operators), captured by the sum $(1\cdot 1 + 0\cdot 0) + (0\cdot 0 + 1\cdot 1) = 1 + 1 = 0$).

Promoting Equation 69 to matrix form, we obtain the following commutation condition.

Concept 3: Arithmetic condition for commutation over $\mathbb{Z}_2$

Let $\left[\Gamma\middle|\Delta\right]$ and $\left[\Pi\middle|\Omega\right]$ be matrices of k and m rows, respectively, which encode the X and Z components of Pauli operators in binary form. Then any two operators

taken from the first and second matrix, respectively, commute iff

$$\sum_{l=1}^{n} \Gamma_{il}\Omega_{jl} + \Delta_{il}\Pi_{jl} = 0 \tag{70}$$

holds $\forall 1 \leq i \leq k, 1 \leq j \leq m$, where addition is performed modulo 2. (cf. [17])

To capture the constraint that any two logical operators commute, set $[\Gamma|\Delta] = [\Pi|\Omega] = [X_L]$ with $[X_L]$ taken from Expression 68, and apply condition 70 to obtain

$$\sum_{l=1}^{k} (U_3)_{il}(V_3)_{jl} + (V_3)_{il}(U_3)_{jl} = 0 \quad \forall 1 \leq i, j \leq k, \tag{71}$$

where other terms cancel due to the zero components in the logical operator form. Expression 71 has an elegant matrix equivalent:

$$U_3 V_3^T + V_3 U_3^T = \mathbf{0}_{k\times k} \tag{72}$$

To satisfy this relation, we follow Gottesman further by setting $U_3 = I, V_3 = 0$. This is a gauge choice, which we later need to take into consideration when constructing the logical Z operator. It reduces our logical operator to the form

$$\left(\begin{array}{ccc|ccc} \overbrace{0}^{r} & \overbrace{U_2}^{s} & \overbrace{\mathbb{1}_k}^{k} & \overbrace{V_1}^{r} & \overbrace{0}^{s} & \overbrace{0}^{k} \end{array}\right). \tag{73}$$

Next, we capture the constraint that every logical operator commutes with every stabiliser. Therefore, we apply Expression 70 to the canonical form 66, contributing its X and Z parts as matrices Γ and Δ, and the logical X matrix 73, contributing its X and Z parts as matrices Π and Ω. Based on Equation 70 again, the commutation constraint can be captured by the two expressions

$$\sum_{l=1}^{r} (\mathbb{1}_r)_{il}(V_1)_{jl} + \sum_{l=1}^{s} (C_1)_{il}(U_2)_{jl} + \sum_{l=1}^{k} (C_2)_{il}(\mathbb{1}_k)_{jl} = 0 \quad \forall 1 \leq i \leq r, 1 \leq j \leq k \tag{74}$$

and

$$\sum_{l=1}^{s} (\mathbb{1}_s)_{il}(U_2)_{jl} + \sum_{l=1}^{s} (E)_{il}(\mathbb{1}_k)_{jl} = 0 \quad \forall 1 \leq i \leq s, 1 \leq j \leq k, \tag{75}$$

corresponding to the first r, and last s rows of 66, respectively. We may express Equations 74 and 75 in matrix form as

$$\mathbf{0}_{(r+s=k)\times k} = \begin{bmatrix} \mathbb{1}_r V_1^T + C_1 U_2^T + C_2 \mathbb{1}_k^T \\ \mathbb{1}_s U_2^T + E\mathbb{1}_k^T \end{bmatrix} = \begin{bmatrix} V_1^T + C_1 U_2^T + C_2 \\ U_2^T + E \end{bmatrix}. \tag{76}$$

Since we wish to determine the remaining variables U_2 and V_1 in Expression 73, we solve the $k \times k$ system 76 for U_2 and V_1, recalling that all additions are performed over $\mathbb{Z}_2$. Then we obtain $U_2 = E^T$ from the lower part, and $V_1 = (C_1 U_2^T + C_2)^T = (C_1 E + C_2)^T = C_2^T + E^T C_1^T$ from the upper part, finally yielding

$$[X_L] = \left[\mathbf{0}_{k\times r} \quad E^T \quad \mathbb{1}_k | E^T C_1^T + C_2^T \quad \mathbf{0}_{k\times(n-r)}\right], \tag{77}$$

which contains our k canonical logical X-operators as the k rows of $2n$ binary entries, explicitly depending on the matrix blocks of the canonical form 66. We see that for the

fully reduced canonical form 24, the first r columns of the Z-part of the logical operator in 77 reduce to C_2^T, as expressed in Matrix 25 (where we denote the $r \times k$ submatrix of the Z-part of the tableau simply as C).

Having determined the standard form of the logical X-operators in Expression 77, we may determine the logical Z-operators in a similar fashion. Lending from Gottesman's notation again, we write

$$[Z_L] = \left(\overbrace{U_1'}^{r} \; \overbrace{U_2'}^{s} \; \overbrace{U_3'}^{k} \;\middle|\; \overbrace{V_1'}^{r} \; \overbrace{V_2'}^{s} \; \overbrace{V_3'}^{k} \right), \tag{78}$$

which we can reduce to

$$[Z_L] = \left(\overbrace{0}^{r} \; \overbrace{U_2'}^{s} \; \overbrace{U_3'}^{k} \;\middle|\; \overbrace{V_1'}^{r} \; \overbrace{0}^{s} \; \overbrace{V_3'}^{k} \right), \tag{79}$$

by reducing by the corresponding stabilisers again. We require that logical Z operators commute pairwise among themselves, yielding the constraint

$$U_3' V_3'^T + V_3' U_3'^T = \mathbf{0}_{k\times k}. \tag{80}$$

We also need to ensure that the i^{th} logical Z-operator anti-commutes with X_{Li}, yet commutes with all X_{Lj} for $j \neq i$, as imposed by the canonical commutation relations.

Similar to Equation 69, the anti-commutation constraint on operators $[\gamma|\delta]$ and $[\pi|\omega]$ is given by

$$\sum_{i=1}^{n} \gamma_i \omega_i + \delta_i \pi_i = 1, \tag{81}$$

where the sum equals 1 instead of 0, caused by an odd number of anti-commuting overlaps.

Using the canonical form of logical X-operators in Expression 73, we then we find that

$$\sum_{l=k}^{n} (V_3')_{il} (\mathbb{1}_k)_{jl} = \begin{cases} 1, & i = j, \\ 0, & \text{otherwise}, \end{cases} \tag{82}$$

which in matrix form reads $V_3' = \mathbb{1}_k$. Since the logical Z-operators also have to commute with all stabilisers, we may further recover the approach from above to write the commutation expressions

$$\sum_{l=1}^{r} (\mathbb{1}_r)_{il} (V_1')_{jl} + \sum_{l=1}^{k} (A_2)_{il} (\mathbb{1}_k)_{jl} + \sum_{l=1}^{s} (C_1)_{il} (U_2')_{jl} + \sum_{l=1}^{k} (C_2)_{il} (U_3')_{jl} = 0 \quad \forall 1 \leq i \leq r, 1 \leq j \leq k \tag{83}$$

and

$$\sum_{l=1}^{s} (\mathbb{1}_s)_{il} (U_2')_{jl} + \sum_{l=1}^{k} (E)_{il} (U_3')_{jl} = 0 \quad \forall 1 \leq i \leq s, 1 \leq j \leq k \tag{84}$$

corresponding to the first r, and last s rows of 66, respectively. Again, we summarise the two expressions in their matrix equivalent

$$\mathbf{0}_{k\times k} = \begin{bmatrix} \mathbb{1}_r V_1'^T + A_2 \mathbb{1}_k^T + C_1 U_2'^T + C_2 U_3'^T \\ \mathbb{1}_s U_2'^T + E U_3'^T \end{bmatrix} = \begin{bmatrix} V_1^T + A_2 + C_1 U_2'^T + C_2 U_3'^T \\ U_2'^T + E U_3'^T \end{bmatrix}. \tag{85}$$

Recall that we want to determine the unknown U_2', U_3', V_1' and V_2' in Expression 79. In order to solve system 85, we therefore apply another gauge choice and set $U_2' = U_3' = 0$, such that the upper block of the matrix constraint yields $V_1 = A_2^T$, whereas the lower one is automatically satisfied. This leaves us with the final form for logical Z operators:

$$[Z_L] = \left[\mathbf{0}_{k\times n} | A_2^T \quad \mathbf{0}_{k\times s} \quad \mathbb{1}_k\right].$$

B.6 Canonicalisation in `QuantumCliiford.jl`

In our simulations, we make heavy use of the `QuantumClifford` [35] library, which exposes the procedures outlined in Section 2.1.6. In particular, it implements the process of determining the Gottesman canonical form of a tableau[124], which is then used to determine logical operators of some code[125]. The latter footnote reference exhibits precisely the same notation that is used in Appendix B.5. For the Gottesman canonicalisation, however, the authors eliminate the C_1-block using the identity block underneath it.

An important remark is in order: In the process of bringing a tableau in its canonical form, we saw that column swaps, i.e., qubit reordering, might need to be performed, which enables us to extract the canonical logical operators. If we then wish to work with the original labeling of qubits, we need to repermute the qubit labels for both stabilisers and logical operators. For example, the canonical form of the Steane code is given by

	X							Z							ϕ
	X_1	X_2	X_3	X_4	X_5	X_6	X_7	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	
S_1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0
S_2	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0
S_3	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0
S_4	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0
S_5	0	0	0	0	0	0	0	1	1	0	0	1	0	1	0
S_6	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0

, (86)

from which we extract the canonical logical operators

$$Z_L = Z_2 Z_3 Z_7 \text{ and } X_L = X_4 X_5 X_7. \tag{87}$$

Here, the rank of the X-submatrix as well as that of the remaining Z block is $r = s = 3$, which yields $k = n - (r+s) = 7 - 6 = 1$ logical qubit, as expected. We see that Tableau 86 indeed exhibits the form of our abstract canonical form 66.

We can create such a tableau automatically via the `MixedDestabilizer` datatype[126] in QuantumSavory, which can be accessed by calling the function

```
MixedDestabilizer(Steane7(), undoperm=false, reportperm=true).
```

[124] For the code implementation, please refer to https://github.com/QuantumSavory/QuantumClifford.jl/blob/master/src/canonicalization.jl#L206-271 (accessed 15.07.2026).

[125] We refer the reader to https://github.com/QuantumSavory/QuantumClifford.jl/blob/master/src/QuantumClifford.jl#L549-645 (accessed 15.07.2026) for the implementation of this procedure.

[126] This datatype is implemented in https://github.com/QuantumSavory/QuantumClifford.jl/blob/4d524965a11b6d8594d578d4935ccd7bc385f56c/src/QuantumClifford.jl#L586 (accessed 15.07.2026).

Suggestively, this call enables us to inspect the resulting permutations of the two canonicalisation steps that were performed:

```
permx = [1, 2, 4, 3, 5, 6, 7], permz = [1, 2, 3, 4, 5, 7, 6]
```

To express the entire permutation as an array, we may use the array composition

```
permx[permz] = [1,2,4,3,5,7,6]
```

This permutation vector indicates which qubits were swapped in the process of canonicalisation. Here, we see that qubits $3 \leftrightarrow 4$ as well as qubits $6 \leftrightarrow 7$ were swapped. If we now want to recover the logical operators corresponding to the original labeling of qubits, we need to perform the inverse permutation of the given permutation, for which we use

```
MixedDestabilizer(Steane7(), undoperm=true),
```

providing us with Tableau 22. Comparing Tableaus 22 and 86, we find that columns $3 \leftrightarrow 4$ and columns $6 \leftrightarrow 7$ were indeed swapped in both the X- and the Z-part of the tableau. Finally, we can extract the canonical logical operators $Z_L = Z_2 Z_4 Z_6$ and $X_L = X_3 X_5 X_6$, corresponding to the initial choice of labeling.

B.7 Codespace Dimension for CSS Codes

Given the CSS code setup in Section 2.1.7, the rank-nullity theorem implies

$$\operatorname{rank}(H_X) = n - \dim(\ker(H_X)) = n - k_x,$$

and likewise

$$\operatorname{rank}(H_Z) = n - \dim(\ker(H_Z)) = n - k_z.$$

This directly implies

$$\operatorname{rank}(H) = \operatorname{rank}(H_X) + \operatorname{rank}(H_Z) = 2n - k_z - k_x = n - k,$$

where $\dim(\ker(H)) = k = k_x + k_z - n$ is precisely the number of logical qubits of the CSS code.

B.8 Commutativity in CSS Code Tableau

Here, we verify the commutativity of the abelian stabiliser group, based on the tableau form in Expression 29. The condition that every row of the r_x rows of H_x commutes with each of the r_z rows of H_z is captured by the equation

$$\sum_{l=1}^{n} (H_X)_{il} (H_Z)_{jl} = 0 \quad \forall 1 \leq i \leq r_x, 1 \leq j \leq r_z \tag{88}$$

as a direct consequence of the arithmetic condition in Equation 70. We can recast this condition as

$$H_X H_Z^T = \mathbf{0}_{r_x \times r_z}. \tag{89}$$

To show that Condition 89 indeed holds for Tableau 29, note that the parity check matrix H of a classical code $\mathcal{C}$ is the so-called *generator matrix* G of its dual code $\mathcal{C}^\perp$, where a generator matrix generates a code via

$$C = \{c' \in \mathbb{F}_q^n : c' = G^T c,\, c \in \mathbb{F}_q^k\}. \tag{90}$$

Therefore, H_X is the generator matrix of $C_X^\perp$, and we can write

$$C_X^\perp = \{c' \in \mathbb{F}_2^n : c' = G_{C_X^\perp}^T c = H_X^T c, c \in \mathbb{F}_2^{r_x}\}.$$

Then for every $c \in \mathbb{F}_2^{r_x}$, we have

$$c^T(H_X H_Z^T) = (c^T H_X) H_Z^T = (H_X^T c)^T H_Z^T = c'^T H_Z^T = (H_Z c')^T = \mathbf{0}_{1\times r_z}, \tag{91}$$

where $c' = H_X^T c$ is a codeword of $C_X^\perp$ and thus also of C_Z (since we have $C_X^\perp \subseteq C_Z$), which implies the last equality by definition. From the chain of equalities 91, it follows that $H_X H_Z^T = \mathbf{0}$, as desired.

B.9 Gottesman Encoding Circuit

The Gottesman circuit for logical basis state encoding is presented in Algorithm 3. In the following, we discuss how to arrive at the standard Gottesman encoding circuit captured by this algorithm. The original formulation by Gottesman [17] turned out to be incorrect in two aspects – the errata were pointed out by Markus Grassl [132] and are documented in [166]. Mondal and Parhi provide a comprehensive step-for-step treatment of encoding circuits [81] based on the orginal Gottesman canonical form, omitting phases. Lastly, the `QuantumClifford` library provides code[127] for constructing an encoding circuit based on the refined canonical form 24, including treatment of phases and final re-permutation of indices, which we discuss in detail below.

A conceptual idea of how to synthesize the logical computational basis state $|c_1 c_2 \dots c_k\rangle$ proceeds in three steps: (i) Start with physical zero state $|0\rangle^{\otimes n}$, (ii) map it to the logical zero state $|0\rangle_L^{\otimes k}$, and (iii) apply the appropriate logical operators $X_{L,1}^{c_1}, \dots, X_{L,k}^{c_k}$ (depending on the logical basis state we wish to create).

Formally, the second step therein (mapping from the physical to the logical all-zero state) entails performing the projection operator

$$\Pi_{\mathcal{S},Z_L} = \Pi_{\mathcal{S}} \Pi_{Z_L} = \prod_{i=1}^{n-k} \frac{\mathbb{1}_{\otimes n} + S_i}{2} \prod_{j=1}^{k} \frac{\mathbb{1}_{\otimes n} + Z_{L,j}}{2}$$

onto the initial state, consistent with the definition that $|0\rangle_L^{\otimes k} = |0\rangle_{L,1} \otimes \dots \otimes X_{L,k}$ is the logical basis state stabilised by the set

$$\mathcal{S} \cup \{Z_{L,j}\}_{j=1}^k.$$

Here, we choose the canonical logical operators 25 and 26 as specific coset representatives for the logical operators, representing the equivalence class of all equivalent logical $Z_{L,j}$ operators, which fix the basis states $|0\rangle_{L,j}$ and $|1\rangle_{L,j}$ of our logical codespace. We may thus write

$$|c_1 c_2 \dots c_k\rangle_L = X_{L,1}^{c_1} \dots X_{L,k}^{c_k} |0\rangle_L = X_{L,1}^{c_1} \dots X_{L,k}^{c_k} (\Pi_{\mathcal{S},Z_L} |0\rangle^{\otimes n}).$$

[127] Please refer to `https://github.com/QuantumSavory/QuantumClifford.jl/blob/master/src/ecc/circuits.jl#L1-83` (accessed 15.07.2026) for the implementation.

Algorithm 3 Logical basis state generation from Gottesman canonical form (based on [17, Ch. 4], adapted from [81])

Input: (i) Quantum register input state $|0\rangle^{\otimes n}$, (ii) Classical k-bitstring $c_1, \dots, c_k$ (computational basis state to encode), (iii) Logical operator matrix $[X_L]$ in canonical form 25, (iv) Canonical Tableau T in fully reduced form 66, (v) Phase information for each stabiliser as $(r+s)$–dimensional vector P

Output: Encoded logical $|c_1 \dots c_k\rangle_L$ state on n qubits

for $i = 1 : k$ **do**
  if $c_i = 1$ **then**
    apply $X(r+s+i)$ ▷ $r+s = n-k$
    for $j = 1 : s$ **do**
      if $[X_L]_{i,r+j} = 1$ **then**
        apply $CX(r+s+i, r+j)$
      end if
    end for
  end if
end for
for $i = 1 : r$ **do**
  apply $H(i)$
  if $T_{i,n+i} = 1$ **then**
    apply $S(i)$
  end if
  for $j = 1 : n, j \neq i$ **do**
    $(x, z) = (T_{i,j}, T_{i,n+j})$
    if $(x, z) = (0, 1)$ & $j \notin [i+1, r+s]$ **then**
      apply $Z - CZ(i, j)$ ▷ Z-controlled Z-gate
    else if $(x, z) = (1, 0)$ **then**
      apply $Z - CX(i, j)$ ▷ Z-controlled X-gate, i.e., CX gate
    else if $(x, z) = (1, 1)$ **then**
      apply $Z - CY(i, j)$ ▷ Z-controlled Y-gate
    end if
  end for
end for
for $i = 1 : (r+s)$ **do**
  if $P_i = -1$ **then**
    if $i \leq r$ **then**
      apply $Z(i)$
    else ▷ $r+1 \leq i \leq n-k$
      apply $X(i)$
    end if
  end if
end for

Now in the derivation of logical operators in Appendix B.5, we saw that the canonical form of the logical Z-operators conveniently consists exclusively of physical Pauli-Z operators. Since we are starting in the computational basis state, we obtain

$$\Big(\prod_{j=1}^{k} \frac{\mathbb{1}_{\otimes n} + Z_{L,j}}{2}\Big) |0\rangle^{\otimes n} = |0\rangle^{\otimes n},$$

which allows us to simplify

$$\Pi_{\mathcal{S},Z_L} |0\rangle^{\otimes n} = \Pi_{\mathcal{S}} |0\rangle^{\otimes n},$$

further implying

$$|c_1 c_2 \dots c_k\rangle_L = X_{L,1}^{c_1} \dots X_{L,k}^{c_k} \Big(\Pi_{\mathcal{S},Z_L} |0\rangle^{\otimes n}\Big) = X_{L,1}^{c_1} \dots X_{L,k}^{c_k} \Big(\Pi_{\mathcal{S}} |0\rangle^{\otimes n}\Big).$$

By construction, the logical X operators commute with the stabilisers, i.e., each logical commutes with all terms in $\Pi_{\mathcal{S}}$. Hence, we may interchange their order to finally obtain

$$|c_1 c_2 \dots c_k\rangle_L = \Pi_{\mathcal{S}} \Big(X_{L,1}^{c_1} \dots X_{L,k}^{c_k} |0\rangle^{\otimes n}\Big).$$

This leaves us with the task of (i*) applying logical X_L operators to the $|0\rangle^{\otimes n}$ state, conditioned on the desired state $|c_1 c_2 \dots c_k\rangle_L$, and (ii*) projecting into the codespace.

For the first step (i*), consider $X_{L,1}^{c_1} \dots X_{L,k}^{c_k} |0\rangle^{\otimes n}$. In order to bring this abstract expression into circuit form, recall the general form of logical X_L operators in Expression 25, where a logical operator in general consists of both X- and Z-components. However, the only possible Z-contribution is on the first r qubits, which don't have any X support; hence the Z-part of the logical operators acts trivially on the physical all-zero state and can be neglected. The X-component of each $X_{L,i}$-operator with $1 \leq i \leq k$ exhibits an X-operator at position $(r+s)+i = (n-k)+i$. Therefore, if we want to implement $c_i = 1$ for $1 \leq i \leq k$, we need to apply the Pauli operator X_{r+s+i}. This transforms the initial state $|00\dots0\rangle$ into $|00\dots0|c_1 c_2 \dots c_k\rangle$, where the binary variables are preceded by $n-k = r+s$ zeros. But note that there is another block of potential non-zero entries in $[X_L]$, denoted by E^T, spanning columns $r+1 : r+s$. For the central insight, if the i^{th} logical operator $X_{L,i}$ is active, i.e., if $c_i = 1$, then the qubit in position $r+s+i$ will be in the $|1\rangle$ state by the action of the $\mathbb{1}_k$ block in $[X_L]$; therefore, applying CX gates from control qubit $r+s+i$ onto all target qubits $r+1 : r+s$ that have support in row i of E^T implements the entirety of logical X_L operators. Note that for any i with $c_i = 0$, adding CXs this way does not incorrectly alter the state, since the control qubit is in the $|0\rangle$ state. This is useful for a general encoding circuit as displayed in Algorithm 4. Applying the initial X_i operators and adding the CXs, we have completed the first step of the encoding. Our quantum register now holds the state

$$|0\rangle^{\otimes r} \otimes |\phi_p\rangle = |\underbrace{00\cdots0}_{r} \underbrace{f_1(c) f_2(c) \cdots f_s(c)}_{s} \underbrace{c_1 c_2 \cdots c_k}_{k}\rangle,$$

where $f_k : \mathbb{F}_2^k \mapsto \{0,1\}$ are functions of the control bits $c = [c_1, c_2, \dots, c_k]^T$ in $|c_1 c_2 \dots c_k\rangle$, and we have denoted $|\phi_p\rangle$ to indicate that this state is still a product state.

For the second step (ii*), we need to construct a circuit that maps into the code space by performing the projection

$$\Pi_{\mathcal{S}} = \prod_{i=1}^{n-k} \frac{\mathbb{1}_{\otimes n} + S_i}{2}.$$

Therefore, consider the canonical tableau containing the respective stabilisers S_i as its rows, and in particular, its upper $r \times 2n$-dimensional matrix block. For the first r qubits, which are in the $|0\rangle$ state, the corresponding matrix blocks are an identity on the X-side, and some matrix B on the Z-side of the tableau. We proceed row for row: To implement the first stabiliser projection, we write (ignoring normalisation)

$$\frac{\mathbb{1}_{\otimes n} + S_1}{2}\left(|0\rangle^{\otimes r} \otimes |\phi_p\rangle\right) \propto |0\rangle^{\otimes r} \otimes |\phi_p\rangle + S_{1,1}\,|0\rangle^{\otimes r} \otimes S_{1,j>r}\,|\phi_p\rangle\,, \tag{92}$$

where the Pauli operator on the first qubit, $S_{1,1}$, is either X or Y (depending on whether or not $B_{1,1} = T_{1,n+1}$ is zero or one), and where we have used that $S_{1,j}$ (the j^{th} Pauli in S_1) is either the identity or Z_j for $2 \leq j \leq r$, and thus fixes the corresponding qubit state $|0\rangle_j$.

Now if $B_{1,1} = 0$, then Expression 92 can be recast as

$$\begin{aligned}\frac{\mathbb{1}_{\otimes n} + S_1}{2}\left(|0\rangle^{\otimes r} \otimes |\phi_p\rangle\right) &\propto |0\rangle^{\otimes r} \otimes |\phi_p\rangle + X_1\,|0\rangle^{\otimes r} \otimes S_{1,j>r}\,|\phi_p\rangle = \\ &= |0\rangle \otimes |0\rangle^{\otimes r-1} \otimes |\phi_p\rangle + |1\rangle \otimes |0\rangle^{\otimes r-1} \otimes S_{1,j>r}\,|\phi_p\rangle\,.\end{aligned} \tag{93}$$

Note that this is precisely what we obtain by applying a Hadamard gate on qubit 1, followed by a controlled $S_{1,j>r}$ gate on the last $n-r$ qubits.

Similarly, if $B_{1,1} = 1$, then Expression 92 can be rewritten as

$$\begin{aligned}\frac{\mathbb{1}_{\otimes n} + S_1}{2}\left(|0\rangle^{\otimes r} \otimes |\phi_p\rangle\right) &\propto |0\rangle^{\otimes r} \otimes |\phi_p\rangle + Y_1\,|0\rangle^{\otimes r} \otimes S_{1,j>r}\,|\phi_p\rangle = \\ &= |0\rangle \otimes |0\rangle^{\otimes r-1} \otimes |\phi_p\rangle + i\,|1\rangle \otimes |0\rangle^{\otimes r-1} \otimes S_{1,j>r}\,|\phi_p\rangle\,.\end{aligned}$$

This, in turn, is what we obtain by applying a Hadamard gate H followed by a phase gate $S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$ on the first qubit, followed again by a controlled $S_{1,j>r}$ gate on the last $n-r$ qubits, since we have

$$SH = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ i & -i \end{bmatrix}.$$

The controlled stabiliser action can readily be described via conditional gates. There are four different cases, corresponding to the Pauli operator that sits at the j^{th} qubit for the stabiliser S_1, with $j > r$. This Pauli operator depends on the exact composition of the submatrices A_1, A_2 and C, of course. For example, if $T_{1,r+2} = 1 = T_{1,n+r+2} \Leftrightarrow S_{1,2} = Y$, then we perform a Z-controlled Y-gate from qubit 1 onto qubit $r+2$. The other three cases are analogous, and all four cases are captured in Algorithm 3.

Having completed the first stabiliser projection via this entangling sequence of Hadamard and Phase gates and controlled operations, we proceed to qubit 2. Since the controlled actions coming from qubit 1 do not target qubits $2:r$, the second qubit will still be in the $|0\rangle$ state, such that we can proceed entirely analogously. The only difference is that we need to include qubit 1 when performing the controlled operations with qubit 2 as control; we have $S_{2,1} \in \{I_1, Z_1\}$ (i.e., the first component of the second stabiliser string is never X, which is true in general for the $S_{i,j}$ components of S_i with $j < i$), but qubit 1 has been transformed via a Hadamard gate and then entangled with other qubits, which implies that Z_1 does not fix it any longer. As one can see in Algorithm 3, we can follow this pattern for the first r qubits: Perform a Hadamard (and

potentially a phase gate) on qubit i with $1 \leq i \leq r$, followed by Z-controlled X-, Y- or Z- operations on all other qubits, where the type of target operation is dictated by the canonical tableau. As a minor detail, one might notice the special range of the indices for the Z-controlled Z-operations, which stems from the B matrix contributing pure Z operators on the first r qubits, acting invariantly on qubits $i+1:r$, as well as from the zero $\mathbf{0}_{r\times s}$ block in the Z-part of the tableau, preventing the possibility of Z-controlled Z-operation for qubits $r+1:r+s$.

This way, we have implemented the first r projections into the codespace, up to the phases we have thus far neglected. Fortunately, inspecting the equivalence in 93, we find that a potential negative phase of stabiliser S_i for $1 \leq i \leq r$ only necessitates applying another Z-gate after the H (or SH) application, since $|0\rangle + (-X_i)|0\rangle = |0\rangle - |1\rangle = Z_i H_i |0\rangle$, and likewise, $|0\rangle + (-Y_i)|0\rangle = |0\rangle - i|1\rangle = Z_i S_i H_i |0\rangle$ hold. But note that all consequent gates involving qubit i are either (Z-)-controlled-Pauli gates controlled by qubit i, or Z-controlled Z gates targeting qubit i. In either case, the post-applied Z_i gate commutes with the gate, such that we can push the phase correction Z-gates to the end of the circuit.

This leaves us with the bottom $(s\times 2n)$-dimensional part of the canonical tableau T, which consists entirely of physical Z operators. To handle this second set of stabiliser projections, recall that these stabilisers commute with all other stabilisers, and importantly also with the logical X_L operators we applied in the beginning. Therefore, we may commute the projectors through to the beginning of the circuit and let them act on the physical all zero state, which it fixes trivially. The only complication arises when some of these stabilisers exhibit a negative phase. If the j^{th} $Z-$type stabiliser ($1 \leq j \leq s$) has a negative phase, we need to apply a corresponding X_{r+j} operation on the initial all-zero state to bring the quantum state in the $+1$-eigenspace of the stabiliser. Note that by doing this, we only affect the qubits $r+1:r:s$. But throughout the entire algorithm, these qubits never act as controls, and only possibly as targets for controlled X operations (as discussed earlier, by the virtue of the $\mathbf{0}_{r\times s}$ matrix in 24, no other controlled-gate is possible in when implementing the first r stabilisers), hence we can move the phase-correction X-gates to the end of the circuit as well (since they act on different qubits, the two types of correction gates commute trivially).

This completes the action of the encoding circuit, and we are left with the logical $|c_1 c_2 \dots c_k\rangle_L$ state of our code. As mentioned above, all steps of the algorithm are summarised in Algorithm 3, and implemented in the `encoding_gott` file of `DQCode` [167]. Note that in order for the encoding circuit to match the original qubit ordering of the stabilisers (which were subject to qubit swaps in the canonicalisation operation), we need to post-apply appropriate $SWAP$ operations, retained from the column swaps producing the tableau standard form (cf. Appendix B.6). In our `Julia` code, we replace this step by relabeling the qubits based on the extracted permutation.

Whereas Algorithm 3 captures the procedure outlined above, the below Algorithm 4 describes the more general version that can be used to encode an arbitrary state $|\psi\rangle$ into its logical counterpart.

Algorithm 4 General encoding circuit from Gottesman canonical form (based on [17, Ch. 4], adapted from [81])

Input: (i) Quantum register input state $|0\rangle^{\otimes n-k} \otimes |\psi\rangle$ on n qubits, with $|\psi\rangle$ the quantum state to encode (ii) Logical operator matrix $[X_L]$ in canonical form 25, (iii) Canonical Tableau T in fully reduced form 66, (iv) Phase information for each stabiliser as $(r+s)$–dimensional vector P

Output: Encoded logical $|\psi\rangle_L$ state on n qubits

for $i = 1 : k$ **do**
  for $j = 1 : s$ **do**
    if $[X_L]_{i,r+j} = 1$ **then**
      apply $CX(r+s+i, r+j)$
    end if
  end for
end for

▷ From here on, follow Algorithm 3

for $i = 1 : r$ **do**
  apply $H(i)$
  if $T_{i,n+i} = 1$ **then**
    apply $S(i)$
  end if
  for $j = 1 : n, j \neq i$ **do**
    $(x, z) = (T_{i,j}, T_{i,n+j})$
    if $(x, z) = (0, 1)$ & $j \notin [i+1, r+s]$ **then**
      apply $Z - CZ(i, j)$
    else if $(x, z) = (1, 0)$ **then**
      apply $Z - CX(i, j)$
    else if $(x, z) = (1, 1)$ **then**
      apply $Z - CY(i, j)$
    end if
  end for
end for
for $i = 1 : (r+s)$ **do**
  if $P_i = -1$ **then**
    if $i \leq r$ **then**
      apply $Z(i)$
    else ▷ $r+1 \leq i \leq n-k$
      apply $X(i)$
    end if
  end if
end for

B.10 Telegate Protocol

Here, we discuss how the telegate protocol depicted in Figure 5 enables the remote execution of a CX gate between two quantum cores, consuming a maximally entangled Bell state. To begin with, our two data qubits q_1 and q_2 are in an arbitrary two-qubit state

$$\alpha|0\rangle_{q_1}\otimes|0\rangle_{q_2}+\beta|0\rangle_{q_1}\otimes|1\rangle_{q_2}+\gamma|1\rangle_{q_1}\otimes|0\rangle_{q_2}+\delta|1\rangle_{q_1}\otimes|1\rangle_{q_2}, \tag{94}$$

where the coefficients satisfy the normalisation condition of quantum states. We wish to perform a CX gate with q_1 as control and q_t as target, i.e., reach the target state

$$\alpha|0\rangle_{q_1}\otimes|0\rangle_{q_2}+\beta|0\rangle_{q_1}\otimes|1\rangle_{q_2}+\gamma|1\rangle_{q_1}\otimes|1\rangle_{q_2}+\delta|1\rangle_{q_1}\otimes|0\rangle_{q_2}. \tag{95}$$

As explained in the main text, we use a shared entanglement resource between the communication qubits $q_{1,\text{comm}}$ and $q_{2,\text{comm}}$ of the two QPUs for this. To this end, assume that we are able to create a noiseless $|\Phi_+\rangle = \frac{1}{\sqrt{2}}(|00\rangle+|11\rangle)$ state between the second and third qubit in Figure 5, which are the respective communication qubits.

The state of the entire 4-qubit system may then be expressed as

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}\Big(\alpha(|0000\rangle+|0110\rangle)+\beta(|0001\rangle+|0111\rangle)+\gamma(|1000\rangle+|1110\rangle)+\delta(|1001\rangle+|1111\rangle)\Big), \tag{96}$$

where the indices correspond to the (tensored) state of the control qubit, communication qubit of QPU 1, communication qubit of QPU 2, and target qubit, respectively.

Next, we apply two local CX gates between the data qubits and the communication qubits, resulting in the overall state

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}\Big(\alpha(|0000\rangle+|0111\rangle)+\beta(|0001\rangle+|0110\rangle)+\gamma(|1100\rangle+|1011\rangle)+\delta(|1101\rangle+|1010\rangle)\Big). \tag{97}$$

At this stage, we measure the first communication qubit in the Z-basis and apply a conditional X operation on the target qubit. In particular, the state of the communication qubit will be $|0\rangle$, in which case no operation is performed on the target qubit, or $|1\rangle$, in which case the X-gate is applied. Note that this conditional gate requires classical communication between the cores, relaying the measurement outcome. For simplicity of notation, we explicitly differentiate the two cases:

$$|\psi_2\rangle_{|q_{1,\text{comm}}\rangle=|0\rangle} = \alpha|0000\rangle+\beta|0001\rangle+\gamma|1011\rangle+\delta|1010\rangle \tag{98}$$

$$|\psi_2\rangle_{|q_{1,\text{comm}}\rangle=|1\rangle} = \alpha|0110\rangle+\beta|0111\rangle+\gamma|1101\rangle+\delta|1100\rangle. \tag{99}$$

Finally, we perform a X-basis measurement on the second communication qubit, followed by conditional gate on the control data qubit. Since we assume access only to Z-basis measurements, we emulate the X-basis measurement by pre-applying a Hadamard gate. This is valid since we have $\langle+|\psi\rangle = \langle0|H^\dagger\psi\rangle = \langle0|H\psi\rangle$, and likewise $\langle-|\psi\rangle = \langle1|H^\dagger\psi\rangle = \langle1|H\psi\rangle$. This yields

$$|\psi_3\rangle_{|q_{1,\text{comm}}\rangle=|0\rangle} = \frac{1}{\sqrt{2}}\Big(\alpha(|0000\rangle+|0010\rangle)+\beta(|0001\rangle+|0011\rangle)+\gamma(|1001\rangle-|1011\rangle)+\delta(|1000\rangle-|1010\rangle)\Big) \tag{100}$$

$$|\psi_3\rangle_{|q_{1,\text{comm}}\rangle=|1\rangle} = \frac{1}{\sqrt{2}}\Big(\alpha(|0100\rangle-|0110\rangle)+\beta(|0101\rangle-|0111\rangle)+\gamma(|1101\rangle+|1111\rangle)+\delta(|1100\rangle+|1110\rangle)\Big), \tag{101}$$

after the application of the Hadamard gates, and finally

$$|\psi_4\rangle_{|q_{1,\text{comm}}\rangle=|0\rangle,|q_{2,\text{comm}}\rangle=|0\rangle} = \alpha|0000\rangle + \beta|0001\rangle + \gamma|1001\rangle + \delta|1000\rangle \tag{102}$$
$$|\psi_4\rangle_{|q_{1,\text{comm}}\rangle=|0\rangle,|q_{2,\text{comm}}\rangle=|1\rangle} = \alpha|0010\rangle + \beta|0011\rangle + \gamma|1011\rangle + \delta|1010\rangle \tag{103}$$
$$|\psi_4\rangle_{|q_{1,\text{comm}}\rangle=|1\rangle,|q_{2,\text{comm}}\rangle=|0\rangle} = \alpha|0100\rangle + \beta|0101\rangle + \gamma|1101\rangle + \delta|1100\rangle \tag{104}$$
$$|\psi_4\rangle_{|q_{1,\text{comm}}\rangle=|1\rangle,|q_{2,\text{comm}}\rangle=|1\rangle} = -\alpha|0110\rangle - \beta|0111\rangle - \gamma|1111\rangle - \delta|1110\rangle\,, \tag{105}$$

after the controlled Z-gate on the first qubit. In each of the four cases, $|\psi_4\rangle$ is a product state of the data qubits q_1 and q_2 and the communication qubits. Since additionally, the latter are fixed by the respective measurement outcomes, we may disregard them, and find that (up to the global phase in the last case), all combinations of measurement outcomes (and their corresponding correction gates) yield that final state

$$|\psi_4\rangle = \alpha|0\rangle_{q_1}\otimes|0\rangle_{q_2} + \beta|0\rangle_{q_1}\otimes|1\rangle_{q_2} + \gamma|1\rangle_{q_1}\otimes|1\rangle_{q_2} + \delta|1\rangle_{q_1}\otimes|0\rangle_{q_2}\,, \tag{106}$$

as desired.

Note that throughout, we assumed the existence of a perfect first Bell state on the communication qubits, which is of course a simplification. In the main text, we discuss how Pauli errors spread through a telegate, and how this affects fault tolerance and logical error rates.

B.11 Stabiliser Tableaus of Selected QEC Codes

In the following, we gather the stabiliser tableaus and qubit mappings utililised in the `DQCode` pipeline.

Steane-[[7,1,3]]

The Gottesman canonical tableau of the Steane code is

```
X_X_X_X
_XX__XX
___XXXX
Z_ZZ_Z_
ZZ__ZZ_
ZZ_Z__Z,
```

with the logical Z-operator $Z_L = Z_2 Z_4 Z_6$.

The seven qubits are mapped via the following assignment

$$\text{QPU } 1 : \{2,3,6,7\}, \qquad \text{QPU } 2 : \{1,4,5\}.$$

Shor-[[9,1,3]]

The Gottesman canonical tableau of the Shor code is

```
XXX___XXX
___XXXXXX
ZZ_______
Z_Z______
___ZZ____
___Z_Z___
______Z_Z
_______ZZ
```

with the logical Z-operator $Z_L = Z_1 Z_4 Z_9$.
The nine qubits are mapped via the following assignments:

$$\text{QPU } 1 : \{1,2,3\}, \qquad \text{QPU } 2 : \{7,8,9\}, \qquad \text{QPU } 3 : \{4,5,6\}.$$

Trivariate Bicycle-[[12,2,3]]

The Gottesman canonical tableau of the Trivariate Bicycle code is

```
X____X___XX_
_X___XX_X___
__X__X_XX_XX
___X_XX_X_XX
____XX_XXXX_
_ZZZ_Z____ZZ
_ZZ___Z____Z
___ZZ__Z___Z
ZZ______Z_Z_
Z_ZZZ____Z_Z
```

with the logical Z-operators $Z_{L,1} = Z_1 Z_3 Z_4 Z_5 Z_{11}$ and $Z_{L,2} = Z_3 Z_4 Z_{12}$.
The unreduced, bounded-weight tableau of the code is (cf. [53])

```
__XX__XX____
X___X__XX___
_X___XX_X___
X____X___XX_
_X_X______XX
__X_X____X_X
Z_Z____Z_Z__
ZZ______Z_Z_
_ZZ___Z____Z
___Z_ZZ___Z_
___ZZ__Z___Z
____ZZ__ZZ__
```

The twelve qubits are mapped via the following assignment:

Four QPUs — QPU 1 : {1,5,10}, QPU 2 : {6,9,11},
QPU 3 : {2,4,7}, QPU 4 : {3,8,12}
Three QPUs — QPU 1 : {2,6,7,11}, QPU 2 : {3,4,8,12}, QPU 3 : {1,5,9,10}
Two QPUs — QPU 1 : {1,5,6,8,9,10}, QPU 2 : {2,3,4,7,11,12}.

Color-[[17,1,5]]

The Gottesman canonical tableau of the color code is

```
X__X_X___X__X_XXX
_X_X________XX___
__XX_X___X___XXXX
____XX______XX___
______X____X__X_X
_______X___X___XX
________XX__XX___
__________XX__XX_
ZZZZ_____________
Z_Z_ZZ___________
Z_Z_____ZZ_______
______ZZ__ZZ_____
_ZZ_Z_ZZZ___Z___Z
ZZ__Z_ZZZ____Z__Z
_______Z__Z___Z_Z
______Z___Z____ZZ
```

with the logical Z-operator $Z_L = Z_1 Z_3 Z_7 Z_8 Z_{17}$.
The 17 qubits are mapped via the following assignments:

QPU 1 : {1,3,5,6,9,10,13,14}, QPU 2 : {2,4,7,8,11,12,15,16,17}.

Bivariate Bicycle-[[18,4,4]]

The Gottesman canonical tableau of the Bivariate Bicycle code is

```
X_______XXXXX__X_X
_X_____XX____X_XX_
__X____X_XXX__X_XX
___X___X__X_XX_XXX
____X___X__X_XXXXX
_____X_XX_XXX_XXXX
______XXXXXX______
_ZZZZZZZ_____ZZ_Z_
__Z___Z_Z_____Z_ZZ
ZZZZZ_Z__Z___Z____
Z__Z__Z___Z__Z__Z_
__ZZ_ZZ____Z_Z__ZZ
ZZZ_________ZZZ___
___ZZZ_________ZZZ
```

with the logical Z-operators $Z_{L,1} = Z_2Z_4Z_5Z_{13}$, $Z_{L,2} = Z_3Z_5Z_6Z_{15}$, $Z_{L,3} = Z_2Z_3Z_4Z_5Z_6Z_{17}$ and $Z_{L,4} = Z_1Z_3Z_4Z_5Z_6Z_{18}$.

The unreduced, bounded-weight tableau of the code is

```
XX_X_____X_X___X__
_XX_X____XX_____X_
X_X__X____XX_____X
___XX_X__X__X_X___
____XX_X__X_XX____
___X_X__X__X_XX___
X_____XX____X__X_X
_X_____XX____X_XX_
__X___X_X_____X_XX
ZZ_Z_____Z_Z___Z__
_ZZ_Z____ZZ_____Z_
Z_Z__Z____ZZ_____Z
___ZZ_Z__Z__Z_Z___
____ZZ_Z__Z_ZZ____
___Z_Z__Z__Z_ZZ___
Z_____ZZ____Z__Z_Z
_Z_____ZZ____Z_ZZ_
__Z___Z_Z_____Z_ZZ
```

The 18 qubits are mapped via the following assignment:

Six QPUs — QPU 1 : {2,9,17}, QPU 2 : {1,3,10}, QPU 3 : {7,15,18},
QPU 4 : {5,11,13}, QPU 5 : {4,6,12}, QPU 6 : {8,14,16}

Three QPUs — QPU 1 : {2,5,8,10,13,16}, QPU 2 : {1,4,7,12,15,18},
QPU 3 : {3,6,9,11,14,17}

Two QPUs — QPU 1 : {3,4,5,6,9,10,12,14,15}, QPU 2 : {1,2,7,8,11,13,16,17,18}.

B.12 DQC Compilation of Gottesman Encoding Circuit

Here, we provide an explanation of Algorithm 2, which reduces the gate count by one for each quartet of suitable CX gates[128]. Any such quartet consists of a pair of two CX gates $\left\{CX_{i\to k}, CX_{j\to k}\right\}$ that share target qubit k and differ in control qubits i and j, as well as a different pair of two CX gates $\left\{CX_{i\to m}, CX_{j\to m}\right\}$, which again share the same target qubit m (with $m \neq k$), and whose control qubits i and j match those of the first pair of CX gates. For the central insight, if between the two pairs of controls on qubits i and j, there are no other gates that map $|0\rangle \Leftrightarrow |1\rangle$ (for example, X- or Y-gates, or the target of another CX gate), then the latter set of controls is active if and only if the first is. This holds, for example, if there are no gates in between these pairs at all, or if we only find controls for other two-qubit gates and Z-type gates. In this case, the target qubit m will be flipped if and only if the target qubit k is flipped by the (isolated) gate action. Hence we may replace the second set of CX-gates with a CX-gate from qubit k to m, which is visualised in Figure 40. For the Gottesman encoding circuits we compile, this condition is initially fulfilled, since qubits $1:r$ only exhibit controls of CX gates. Furthermore, performing only one compilation pass assures that we never violate the constraint, since new targets of two-qubit gates are only added on qubits $r+1:n$[129].

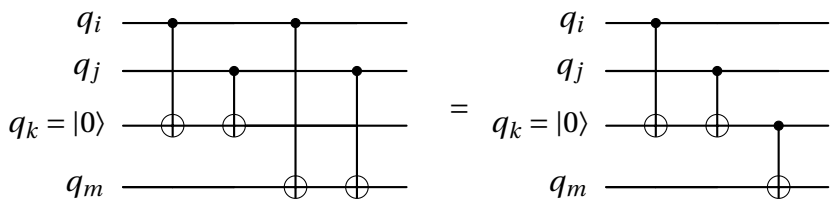


Figure 40: Reducing the gate count of a CX circuit by exploiting the overlap between gates [80]. We require qubit q_k to be in the $|0\rangle$ state, whereas the subsystem of the other three qubits starts is in some arbitrary state $|\psi\rangle$.

However, there is an important constraint: Since we utilise qubit k as a control qubit, it must not be affected by previous gate action. As depicted in Figure 40, we require $q_k = |0\rangle$ to guarantee that the gate reduction in Figure 40 does not alter the overall unitary on the given input state. By the structure of the Gottesman encoding circuit for the logical $|0\rangle_L^{\otimes k}$ state of some CSS code, the initial target qubits of Algorithm 1 are among qubits $r+1:n$ and not affected by any other gate action, since we start in the physical $|0\rangle^{\otimes n}$ state, and CX gates are applied from qubits $1:r$ onto $r+1:n$ exclusively. Therefore, performing only one compilation pass, in which we additionally flag the qubits that have already participated in compilation by being part of a quartet,

[128] Please refer to `https://github.com/timneumann1/DQCode/blob/main/src/encoding_gott.jl#L176-308` for a `Julia` implementation of the algorithm.

[129] Note that the qubit indices used in the previous sentences originate from the canonicalised tableau, which potentially includes qubit permutations. This simplifies reasoning, even though we apply the DQC compilation technique in Algorithm 2 on the original qubit ordering; for this original ordering, the control and target qubits will still be cleanly separated (yet not in the convenient ordering from above), which is why the above argument as well as the following ones are valid also for the original ordering of qubits.

we ensure that this additional constraint is fulfilled. Throughout, we insert the new (triplet) gate group right at the next available spot in the compiled circuit, which we thus build triplet by triplet, as captured by Algorithm 2. Gates that are not part of any quartet only have a target on qubits $r+1:n$ by construction, and hence can be commuted through to the end of the circuit[130].

We employ this compilation method to greedily find CX quartets in the Gottesman encoding circuit, reducing the effective gate count. In the DQC setting, however, we do not aim at reducing the gate count unconditionally; instead, our primary objective is to reduce *tele*gates. In the steps we have outlined so far, there is one special case in which the above method actually *increases* the telegate count (while decreasing the overall gate count): Let qubits i, j and m share the same core, whereas qubit k inhabits a different core. Then adding a telegate between qubits k and m at the 'benefit' of deleting two intra-core two-qubit gates is not a desirable compilation step. Fortunately, since the CX gates commute, we can adapt our protocol slightly by inserting a telegate from m to k (i.e., with flipped control and target) in this case, while deleting the *two* telegates $CX(i,k)$ and $CX(j,k)$. This yields to a reduction of the telegate count by one.

The proposed DQC-aware overlap compilation algorithm rests on many assumptions on circuit structure, since it is tailored specifically for logical zero state encoding. For this use case, it achieves a respectable decrease in telegate count by means of analytic circuit compilation. These results can be inspected in Figure 15 for multiple QEC codes and Type-II architectures.

B.13 Non-FT State Preparation Circuits for Selected QEC Codes

Here, we collect all state preparation circuits discovered by the Genetic Algorithm, as used for the DQC simulations in Section 5. Each circuit is optimised for the DQC setting based on the qubit mapping captured in Appendix B.11.

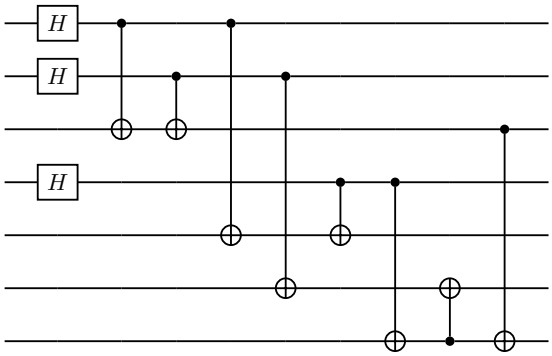


Figure 41: Non-FT state preparation circuit synthesising the logical $|0\rangle_L$ state of the Steane code on two cores.

[130] In the last paragraph, indices again referred to the canonicalised circuit, and the argument proceeding entirely analogously for the restored original ordering of qubits in the actual encoding circuit that we compile.

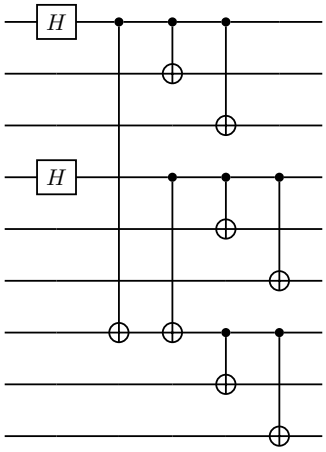


Figure 42: Non-FT state preparation circuit synthesising the logical $|0\rangle_L$ state of the Shor code on two cores.

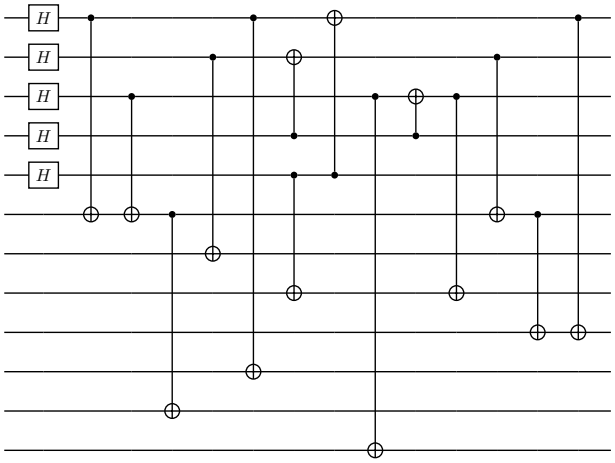


Figure 43: Non-FT state preparation circuit synthesising the logical $|00\rangle_L$ state of the Trivariate Bicycle-$[[12,2,3]]$ code on four cores.

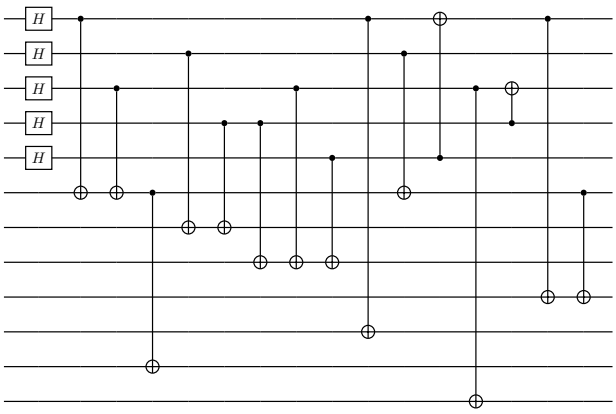


Figure 44: Non-FT state preparation circuit synthesising the logical $|00\rangle_L$ state of the Trivariate Bicycle-$[[12,2,3]]$ code on three cores.

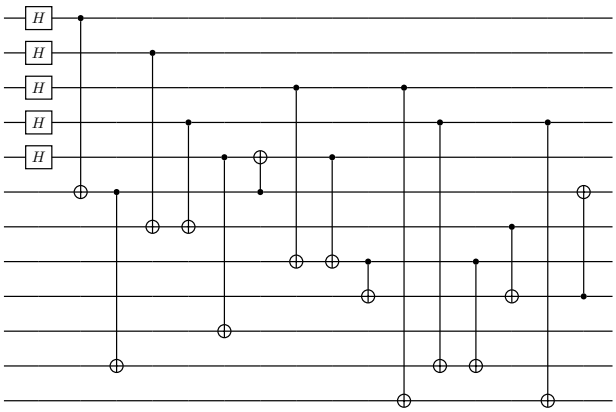


Figure 45: Non-FT state preparation circuit synthesising the logical $|00\rangle_L$ state of the Trivariate Bicycle-$[[12,2,3]]$ code on two cores.

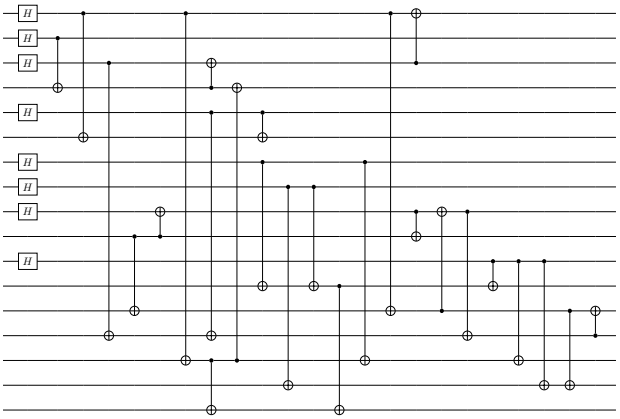


Figure 46: Non-FT state preparation circuit synthesising the logical $|0\rangle_L$ state of the Color-$[[17,1,5]]$ code on two cores.

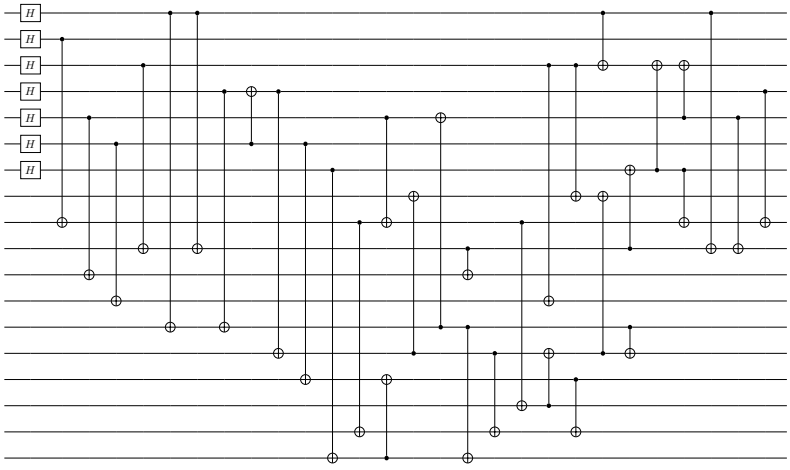


Figure 47: Non-FT state preparation circuit synthesising the logical $|0000\rangle_L$ state of the Bivariate Bicycle-$[[18,4,4]]$ code on six cores.

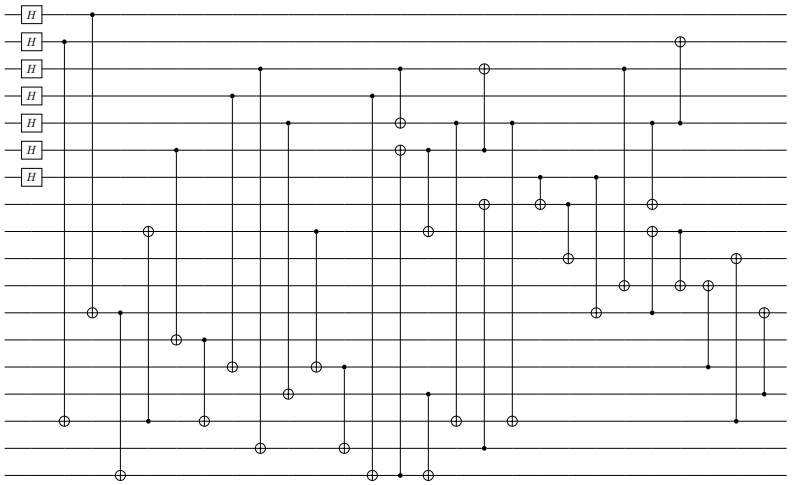


Figure 48: Non-FT state preparation circuit synthesising the logical $|0000\rangle_L$ state of the Bivariate Bicycle-$[[18,4,4]]$ code on three cores.

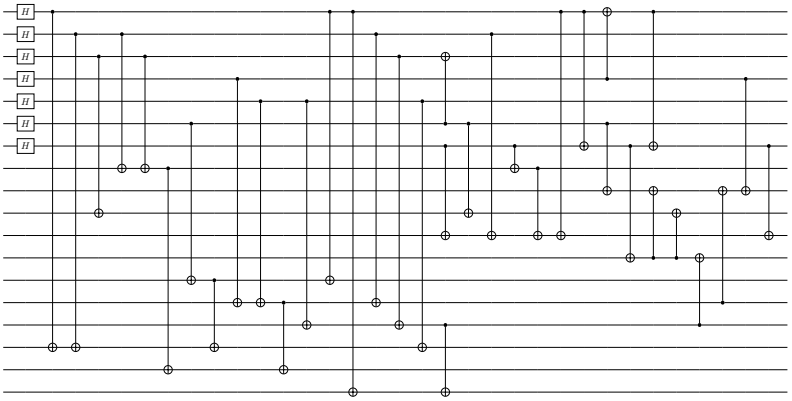


Figure 49: Non-FT state preparation circuit synthesising the logical $|0000\rangle_L$ state of the Bivariate Bicycle-$[[18,4,4]]$ code on two cores.

B.14 From Markov Decision Process to Monte Carlo Tree Search

In this appendix, we discuss the mathematical underpinnings of performing a Monte Carlo Tree Search for our encoding circuit search problem. For this, it is instructive to phrase the problem as a *Markov Decision Process*, which models sequential decision making under uncertainty.

Definition 11: Markov Decision Processes

A Markov Decision Process (MDP) is a tuple (S, A, R), where S is the state space of the environment, $A(s)$ is the space of actions an agent can take in state $s \in S$, and $R : (S \times A \times S) \to \mathbb{R}$ is the reward function, yielding the reward $R(s, a, s')$ of moving from state $s \in S$ to state $s' \in S$ by applying action $a \in A$. The policy $\pi : S \to A$ determines which action $a \in A$ the agent takes in state $s \in S$. (cf. [168])

Many flavours of Markov Decision Processes exist; for example, one can define continuous or discrete action and state spaces, and discounted MDPs in which rewards of future actions contribute to the value assigned to the current state.

In this work, we define the state space to be the (discrete) set of all possible encoding circuits according to the $H-CX$ template discussed earlier, with the initial state being an empty circuit, and the implicit assumption that each circuit is applied onto the physical all-zero state. Furthermore, we define the action space to be the set of quantum gates that can be added to a given partial encoding circuit. In particular, if $s \in S$ is some state (i.e., some circuit), for which the fixed number of Hadamard gates in the template is not yet exhausted, $A(s)$ consists of all Hadamard gates that have not yet been added to the circuit. Otherwise, i.e., if the Hadamard layer is completed, $A(s)$ consists of all CX gates. In the latter case, we restrict the action space by adding the constraint that control qubits can only be those that already experienced a Hadamard, or the target of another CX gate, in order to avoid inactive two-qubit gates. Since we wish to find efficient encoding circuits, we add a discount factor γ that incentivises shorter circuits. Commonly, MDPs build upon probabilistic transition functions: Performing action a in state s maps to a new state s' with a certain probability $p_{s,s'}(a)$, such that $\sum_{s' \in S} p_{s,s'}(a) = 1$ holds [169, Section 1.2.1]. We relax this concept by assuming deterministic transitions, where a given state (circuit) and action (gate) uniquely determines the next state (the circuit with the gate appended). This simplification still respects the Markov property, since the transition does only depend on s and a, and not on the history of the process. Choosing this special case conceptually settles our optimisation problem somewhere in between MDPs and deterministic search; nonetheless, we follow the framework of (deterministic) MDPs as it provides a clear motivation of *Monte Carlo Tree Search* (MCTS) as our search method. In a more general MDP framework, one could also make rewards stochastic, which we circumvent by simulating our circuits in a noiseless setting. And finally, the concept of *Partially Observable Markov Decision Processes* introduces stochasticity to the state space, where the agent cannot fully determine the current state; we mitigate such circumstance via Clifford simulation, in which the agent can retrieve any state (circuit) efficiently via its equivalence with a stabiliser tableau (we use the canonical tableau here as well).

Casting MDPs as decision making models, the task of determining an optimal decision enters the picture via the policy π. Here, we focus on a deterministic policy $\pi : S \to A$, rather than a stochastic one $\pi : S \times A \to \mathbb{R}$ [168]. This does by no means im-

ply that for a given state s, the action space $A(s)$ is trivial; rather, the policy extracted from the agent interacting with the environment indicates the unique action following the observation of a given state. Generally, we are interested in choosing π that optimises the total expected (discounted) reward $V^\pi(s_0)$, where V is a value function on some initial (i.e., current) state s_0 of the environment. Given some discount factor γ, we write

$$\begin{aligned} V^\pi(s_0) &= \sum_{t\geq 0} \gamma^t R(s_t, a_t = \pi(s_t), s_{t+1}(s_t, a_t)) = R(s_0, a_0, s_1) + \sum_{t\geq 1} \gamma^t R(s_t, a_t, s_{t+1}) \\ &= R(s_0, a_0, s_1) + \gamma \sum_{t\geq 0} \gamma^t R(s_{t+1}, a_{t+1}, s_{t+2}) = R(s_0, a_0, s_1) + \gamma V^\pi(s_1). \end{aligned} \tag{107}$$

As Equation 107 shows, we have expressed the value function of some initial state s_0 in terms of its immediate reward and the (discounted) value function of state s_1. When working with a stochastic policy and/or transition function, we need to introduce expectation values over possible actions (under the stochastic policy) and next states (under the stochastic transition), which is the formulation often encountered in literature [169].

Denote the optimal value of state s_i as $V(s_i) = \max_\pi V^\pi(s_i)$. Then for a given state $s_0 \in S$, the optimisation task at hand is to find an optimal action $a_0 = \pi^*(s_0)$ that solves

$$V(s_0) = \max_{a_0 = \pi(s_0) \in A(s_0)} \Big(R(s_0, a_0, s_1(s_0, a_0)) + \gamma V(s_1) \Big), \tag{108}$$

i.e.,

$$\pi^*(s_0) = \arg\max_{a_0} \Big(R(s_0, a_0, s_1) + \gamma V(s_1) \Big), \tag{109}$$

where the term $V(s_1)$ is obtained from following some optimal policy π^* after performing action a_0 and reaching state s_1. Within our deterministic setting, Equation 108 is referred to as the *Bellman equation*, providing a necessary condition for an optimal policy π^* (whereas in the usual stochastic setting of MDPs, the Bellman equation is the optimum of an expectation value). Since $\pi^*(s_t) = a_t$ prescribes the next action to be taken from state s_t, it is clear that finding the optimal policy amounts to determining the next best action to take in a process of sequential decision making. In this context, it is common to define the so-called *Q-value* of a state-action pair as

$$Q(s_0, a_0) = R(s_0, a_0, s_1) + \gamma V(s_1), \tag{110}$$

which captures the quality of choosing some action a in state s via $\pi(s) = a$ (and following some optimal policy thereafter). Using this, we may equivalently express Equation 108 as

$$V(s_0) = \max_{a_0 \in A(s_0)} Q(s_0, a_0), \tag{111}$$

and Equation 109 as

$$\pi^*(s_0) = \arg\max_{a_0} Q(s_0, a_0) \tag{112}$$

where we again avoid expectation values over actions and next states by working with a deterministic policy and transition. The Q-value features prominently in many modern Reinforcement Learning techniques, such as *Q-Learning*. For example, already one decade ago, Google DeepMind used a deep Q-network (a type of deep neural network) for training agents to play Atari games [170].

As pointed out in [168], we can classify two main categories of optimisation methods for MDPs – value-based and policy-based methods. While the latter have gained increased attention, e.g., in Actor-Critic RL, in this work we focus on value-based methods. Therefore, consider a conceptually straightforward method of using the Bellman equation for obtaining an optimal policy: Assign some initial value to each state, and iteratively compute the value of each state according to Equation 108 by updating the value of each state s with the largest current Q-value from Equation 110 over all possible actions $a \in A(s)$. This procedure converges [171, Ch. 3.5], yielding the value function for each state, $V(s)$, as the maximum over all final Q-values $Q(s,a)$ of state-action pairs. At this point, the optimal policy for state s can be extracted as the largest Q-value over all actions, according to Equation 112.

However, for many problems, the state space S is too large to be traversed. In this work, the action space typically contains a number of gates on the order of $\mathcal{O}(n^2)$ for n qubits, but the state space (the number of possible stabiliser tableaus) grows exponentially in n^2 (cf. Section 2.1), which is intractably large even for small instances of QEC codes. A remedy to this predicament is to merely approximate the Q-value for each state-action pair. For example, [170] uses a neural network as function approximator to achieve this task. As an alternative, in this work we utilise Monte Carlo Tree Search (MCTS) [172] for our search of efficient encoding circuits, which works by building a tree of states and actions via repeated (Monte Carlo) sampling.

B.15 Monte Carlo Tree Search: Remarks

This section contains a number of remarks about the Monte Carlo Tree Search implemented in `DQCode`. These remarks are not strictly necessary to understand the general working principle of the algorithm, but are in order to contextualise the mathematical background detailed in Appendix B.14.

- (i) Two special cases exist in which some phases of the algorithm are skipped: If a pre-specified depth (a hyperparameter) is reached, or if a terminal state (zero tableau distance) is encountered. In the former case, the reward has already been determined, and we merely update the node counts in the fourth phase; in the latter case, the search is terminated entirely, since we have encountered a valid encoding circuit;

- (ii) Having determined the optimal action (gate) to be performed based on the Q-value approximation via Monte Carlo sampling, we construct a new search tree, starting from the optimal state resulting from applying the action to the previous optimal state.

- (iii) For each of these tree builds in this sequential decision making process, selection is performed based on a rule, the *Upper Confidence Bound applied to Trees* (UCT) [173]. Given some node for state s, this rule selects the node for action a that maximises the expression

$$\mathrm{UCT}(s,a) = Q(s,a) + c\sqrt{\frac{\log(N(s))}{N(s,a)}}. \tag{113}$$

  Here, $Q(s,a)$ is the current Q-value of the a node originating from the s node, $N(s)$ counts the number of visits to the state node, and $N(s,a)$ counts the number of visits to the derived action node. If some action node is encountered for

the first time, it is directly selected, such that no division-by-zero error occurs. In Expression 113, the first term controls the exploitation of knowledge about the Q-value of a state-action pair, where the second term adds an exploration component to the search. The hyperparameter c controls the relative weighting of the two terms. Having obtained this a, we also add the next state node s' deterministically; if our MDP was stochastic, we would randomly sample the node from all possible next states given s and a.

- (iv) There are multiple hyperparameters that guide the MCTS. Besides from the aforementioned exploration constant c, we also specify a depth up to which the tree is traversed, and a number of iterations indicating how many samples are taken. The interplay of these parameters defines intricate trade-offs. For example, for a given exploration constant or depth, we need sufficiently many iterations in order to not only explore but also exploit the space effectively, while maintaining reasonable runtime. In many cases, a lower depth led to better results, which we attribute to a failure to exploit the search space with a constant number of Monte Carlo samples for large depth. Furthermore, choosing to reuse the search tree from one step of the search to another can accelerate the algorithm, since some paths beyond the new root node have already been simulated (given large enough depth). However, this reuse also imposes a large memory overhead.

- (v) The updates of the backpropagation algorithm alter the quantities of the UCT function from above[131]. In particular, we traverse backwards from the leaf node to the root, increasing the counts $N(s)$ and $N(s,a)$ by one at each node along the way. Furthermore, we update the Q-value proxies on this decision process path as $Q(s,a) \leftarrow Q(s,a) + \frac{q-Q(s,a)}{N(s,a)}$, where $q = r(s,a,s') + \gamma V$ is the sum of the reward $r(s,a,s')$ (48) obtained from simulating node s' and the discounted term γV, where V approximates the value function $V(s')$. For leaf nodes, V is set to 0; afterwards, it is being passed towards the root node in discounted form $V \leftarrow r(s,a,s') + \gamma V$ at each step.

- (vi) There are two stopping conditions: Firstly, when the search reaches a terminal state, and secondly, when a pre-allocated maximal number of steps has been taken (i.e., the circuit length surpasses some threshold).

- (vii) Once the algorithm has terminated, we can extract the (approximated) optimal policy based on the approximated Q-values. In particular, we start with the root node (the empty circuit) and sequentially select the action a that maximise $Q(s,a)$ for a given state s. This way, we can build our circuit gate-by-gate, from scratch.

B.16 State Preparation and Measurement Error

Here, we provide a brief justification of treating the SPAM error reported in Table 2 via two separate and equi-probable errors in our DQC noise model (cf. Table 3). In [160], the SPAM error is reported with one value, since in a real benchmarking experiment,

[131] The code implementation of this can be found in https://github.com/JuliaPOMDP/MCTS.jl/blob/master/src/vanilla.jl#240-276.

it can be difficult to dissect whether errors stem from the preparation or measurement phase. If we assume that state preparation and measurement separately contribute equi-probable errors p_{sm}, where a SPAM error probability p is registered if *precisely one* of the routines is erroneous (such that errors don't cancel out), we have

$$p = 2p_{sm}(1-p_{sm}). \tag{114}$$

Solving this for p_{sm} yields

$$p_{sm} = \frac{1}{2} \pm \frac{\sqrt{1-2p}}{2} \approx \frac{1 \pm (1-p)}{2}, \tag{115}$$

where we have approximated the square root with a first-order Taylor expansion around $p = 0$, assuming that $p << 1$. Under this assumption, we disregard the solution $p_{sm} = 1-p/2$ and keep $p_{sm} = p/2$. We see that that p_{sm} and p are on the same order of magnitude in the low-noise regime. We then set $p_{sm} = p$ in order to align the two SPAM error types with the two-qubit gate error, which provides better insight into how physical and logical error rates relate.

B.17 Idling Error Probability Per Telegate Round

In the main text, we introduce the concept of a telegate round, for which we model decoherence of idle qubits by applying $\eta = 12$ rounds of single-qubit noise sampled from a depolarising channel with error probability $p_{\text{idle}} = p/10$ each. In order to improve the efficiency of our DQC simulation, we seek to approximate the resulting idling error probability $p_{\text{idle}_{\text{tele}}}$ up to first order, necessitating only one sample of the depolarising noise channel (with adjusted error probability $p_{\text{idle}_{\text{tele}}}$).

To obtain such an approximation, we first consider the true probability $p_{\text{idle}_{\text{tele}}}$. Since errors can cancel out, this probability is given by

$$p_{\text{idle}_{\text{tele}}} = \eta p_{\text{idle}}(1-p_{\text{idle}})^{\eta-1} + \mathcal{O}(p^2) = \eta p_{\text{idle}}\Big(1 - \eta p_{\text{idle}} + \mathcal{O}(p^2)\Big) + \mathcal{O}(p^2) = \eta p_{\text{idle}} + \mathcal{O}(p^2). \tag{116}$$

Now consider

$$1-(1-p_{\text{idle}})^{\eta} = 1 - \Big(1 - \eta p_{\text{idle}} + \mathcal{O}(p^2)\Big) = \eta p_{\text{idle}} + \mathcal{O}(p^2), \tag{117}$$

which agrees with $p_{\text{idle}_{\text{tele}}}$ up to first order.

Thus, when sampling one depolarising noise channel of error probability $p_{\text{idle}_{\text{tele}}}$, we may approximate $p_{\text{idle}_{\text{tele}}} \approx 1-(1-p_{\text{idle}})^{\eta}$ up to first order in p (and p_{idle}).

B.18 LIER and Logical Z-measurements

In the following, we provide some more background on Equation 59 and its consequences for our simulations. In particular, we firstly develop a theory of LIER for more general noise channels (potentially resulting in more general logical states prior to logical $Z_{L,i}$-measurement), and then relate it back to the simplified (yet still expressive) case of logical Pauli errors from depolarising noise.

So assume that we have some (pure) quantum state $|\psi_c\rangle_L^{(s)}$ after error correction[132]. This state is certainly a logical state, yet not necessarily a product state. We may still

[132]While an arbitrary noise channel will in general lead to a classical mixture after averaging over all possible errors, we are only considering a single trajectory with noise sampled from the noise channel here.

estimate the logical initialisation error rate of the i^{th} logical qubit according to

$$\overline{\text{LIER}_i} = \frac{1}{\#_{\text{acc}}} \sum_{s=1}^{\#_{\text{acc}}} \mathbb{P}\Big(|\psi_c\rangle_{L,i}^{(s)} = |1\rangle_{L,i} \Big), \tag{118}$$

as the probability of finding the i^{th} logical qubit in the $|1\rangle_{L,i}$ state. However, this probability is now not binary, but depends on the decomposition

$$|\psi_c\rangle_L^{(s)} = \alpha\, |0\rangle_{L,i} \otimes |\psi'\rangle + \beta\, |1\rangle_{L,i} \otimes |\psi''\rangle\,, \tag{119}$$

where $|\psi'\rangle$ and $|\psi''\rangle$ are the normalised components of the remaining system (such that α and β fully determine the overlap of $|\psi_c\rangle_L$ with the $|0\rangle_{L,i}$ and $|1\rangle_{L,i}$). Then we have

$$\mathbb{P}\Big(|\psi_c\rangle_{L,i}^{(s)} = |1\rangle_{L,i} \Big) = \Big\| \Big(\langle 1|_{L,i} \otimes \mathbb{1}_{j\neq i} \Big) |\psi_c\rangle_L \Big\|^2 = \Big\| \beta\, |\psi''\rangle \Big\|^2 = |\beta|^2. \tag{120}$$

Here, the probability of obtaining $|0\rangle_{L,i}$ by measuring $Z_{L,i}$ is β^2, according to Born's rule.

Next, consider the expectation value

$$\Big\langle \psi_c^s \Big|_{L,i} \Big(\frac{1 - Z_{L,i}}{2} \Big) \otimes \mathbb{1}_{L,j\neq i} \Big| \psi_c^s \Big\rangle_{L,i} = |\beta|^2 \cdot \Big\langle 1 \Big|_{L,i} \Big(\frac{1 - Z_{L,i}}{2} \Big) \Big| 1 \Big\rangle_{L,i} \cdot \Big\langle \psi'' \Big| \psi'' \Big\rangle = |\beta|^2,$$

where the identity operator acts on all logical qubits other than the i^{th}, the first equality follows from expanding the logical operator and applying the projection operator $\frac{1-Z_{L,i}}{2}$, and the last equality follows from both inner products evaluating to 1. Thus, we can use the expectation value to estimate the logical initialisation error rate,

$$\overline{\text{LIER}_i} = \frac{1}{\#_{\text{acc}}} \sum_{s=1}^{\#_{\text{acc}}} \Big\langle \psi_c^s \Big|_{L,i} \Big(\frac{1 - Z_{L,i}}{2} \Big) \Big| \psi_c^s \Big\rangle_{L,i}. \tag{121}$$

Note that here we have not made any assumption of each $|\psi\rangle_{L,i}$ being exclusively $|0\rangle_{L,i}$ or $|1\rangle_{L,i}$; therefore, this formalism also applies to an experiment with a more general error channel, which might induce logical non-Pauli errors. However, we are facing a complication: while the expectation value of the logical Z-projector on the corrected logical state yields the correct probability, in practice we need to approximate this probability via many samples; this is conceptually the same as performing multiple samples to obtain an estimator for the LIER_i, yet it requires multiple samples for *each individual* corrected state. The reason for this is that we are still working with a sequence of Bernoulli random variables, but only *after* measurements, which adds additional variance.

In our work, we are using a depolarising channel. Since every (possibly coherent) error can be expressed as a superposition of Pauli strings, this is actually quite expressive in terms of testing error-correcting capabilities. As stated in [3, Ch. 10.3.1], "the ability to error-correct the depolari[s]ing channel automatically implies the ability to error-correct an *arbitrary* single qubit quantum operation". However, in terms of uncorrectable errors, the depolarising channel can only lead to a logical Pauli error for each sample, whereas for other error channels more general logical errors could be mimicked[133]. As discussed in the main text, this greatly simplifies matters; each measurement for the expectation value

$$\Big\langle \psi_c^s \Big|_{L,i} \Big(\frac{1 - Z_{L,i}}{2} \Big) \Big| \psi_c^s \Big\rangle_{L,i} \tag{122}$$

[133] Even though logical non-Pauli errors probably rarely occur by random noise, since they require very particular physical errors of high weight.

is now deterministic[134]. As a consequence, the expectation value of the logical $Z_{L,i}$ observable can be obtained via a single measurement, eliminating the stochasticity of measurement.[135] Scaling the measurement outcome $m = \pm 1$ via $(1-m)/2$ adapts the outcome to the formalism of the Bernoulli trials discussed throughout.

B.19 Standard Deviation of $\overline{\text{LIER}_i}$

Here, we derive the standard deviation of our estimator $\overline{\text{LIER}_i}$, based on the formalism in Section 5.2.4.

Since X_i is a Bernoulli random variable, we have $\text{Var}[X_i] = \text{LIER}_i(1-\text{LIER}_i)$, and

$$\text{Var}\left[\overline{\text{LIER}_i}\right] = \text{Var}\left[\frac{1}{n_{\text{acc}}}\sum_{s=1}^{n_{\text{acc}}} X_i^{(s)}\right] = \frac{1}{(n_{\text{acc}})^2}\sum_{s=1}^{n_{\text{acc}}}\text{Var}[X_i^{(s)}] = \tag{123}$$

$$\frac{n_{\text{acc}}}{(n_{\text{acc}})^2}\text{Var}[X_i^{(s)}] = \frac{1}{n_{\text{acc}}}\text{LIER}_i(1-\text{LIER}_i), \tag{124}$$

where we have used that the random variables are independent and identically distributed. Since we don't have access to the true value of LIER_i, we substitute our sample mean to obtain the final equation

$$\text{Var}\left[\overline{\text{LIER}_i}\right] = \frac{\overline{\text{LIER}_i}(1-\overline{\text{LIER}_i})}{n_{\text{acc}}}. \tag{125}$$

For a large number of accepted runs, the Central Limit Theorem asserts that

$$\frac{\overline{\text{LIER}_i} - \text{LIER}_i}{\sqrt{\text{Var}[X_i]/n_{\text{acc}}}} \sim \mathcal{N}(0,1)$$

follows a standard normal distribution, which is why we plot standard deviations

$$\sigma\left[\overline{\text{LIER}_i}\right] = \sqrt{\frac{\overline{\text{LIER}_i}(1-\overline{\text{LIER}_i})}{n_{\text{acc}}}}$$

symmetrically around our estimates $\overline{\text{LIER}_i}$, similar to [44, Figure 3].

Also for the acceptance rate AR, the underlying random variable is assumed to be Bernoulli-distributed with acceptance probability AR, so the sample mean $\overline{\text{AR}}$, estimating the true underlying acceptance rate of the encoding routine, has variance

$$\text{Var}[\overline{\text{AR}}] = \frac{\overline{\text{AR}}(1-\overline{\text{AR}})}{n_{\text{traj}}}. \tag{126}$$

[134] I.e., measurement deterministically projects onto the eigenspace corresponding to the eigenvalue -1 of the logical operator $Z_{L,i}$ (the undesired logical $|1\rangle_i$ state) with binary probability (0 or 1) given in the second term of Equation 59, again conditioned on the run being accepted.

[135] From an epistemological perspective, this eliminates a source uncertainty in our DQC simulations. In any real experiment, in addition to the probabilistic nature of the error channel (mimicked by Monte Carlo sampling of errors in the simulation, leading to different $|\psi_c\rangle$ states), measurement expectation values as in Equation 59 can in general only be approximated via repeated measurements (since any single measurement of some Hermitian operator collapses the state to an eigenstate of the operator with a certain probability, which we can only approximate at a certain resolution); here, all stochasticity comes from the sampling of noise, for which we require a large amount of samples to reduce the variance of our estimator.

Based on the construction of $\overline{\text{LIER}}$ as the average of individual estimators in Definition 10, we can also estimate $\text{Var}[\overline{\text{LIER}}]$. When reporting this variance, we make the simplifying assumption that all LIER_i are independent, with the sample means $\overline{\text{LIER}_i}$ and their variances comparable in magnitude, such that we can write

$$\text{Var}[\overline{\text{LIER}}] = \text{Var}\Big[\frac{1}{k}\sum_{i=1}^{k}\overline{\text{LIER}_i}\Big] = \frac{1}{k^2}\sum_{i=1}^{k}\text{Var}[\overline{\text{LIER}_i}] = \frac{\overline{\text{LIER}}(1-\overline{\text{LIER}})}{kn_{\text{acc}}}. \tag{127}$$

B.20 DQC Simulation: Mapping and Operations

Here, we discuss some aspects of the qubit mapping and execution of operations for our DQC simulations in greater detail.

As visualised in Figure 24, in order to perform operations between QPUs, we map data qubits onto their respective cores via SWAP gates at the beginning of the circuit. In Section 4, we executed circuits on the lexicograpical ordering and kept track of the DQC mapping separately for telegate counts; here, we perform this mapping for visual insight. In a laboratory setting, this mapping according to the partitioning identified in 4.1.3 can be achieved by simply relabeling all gate operations and tracking the ordering of qubits for syndrome extraction and measurement (without a physical process of mapping), hence SWAP gates are simulated noiselessly. Note that we only visualise the mapping of data qubits; the communication qubits are appended below the data qubits, where each group of $n_{\text{cores}}-1$ belongs to a separate QPU, and for the ancilla qubits (displayed below the communication qubits), we virtually keep track of QPU membership (which is determined based on the core that each ancilla interacts with the most).

For the Trivariate-Bicycle code on two cores, the QEC-cycle-aware mapping (visualised in Figure 8) is captured by the vector

$$[\pi(1),\pi(2),\ldots,\pi(n)] = [1,5,6,8,9,10,2,3,4,7,11,12],$$

where the permutation $\pi : Q \mapsto Q$ (a bijective map from the set of all qubits Q to itself) indicates which data qubit is located at which DQC position — the j^{th} qubit sits at DQC index i^{th} by $\pi(i) = j$[136]. That means, the first six qubits in the list can be found in the first QPU, with the remaining six located in the second QPU. The connection to the SWAPs in Figure 24 is made by expressing the permutation captured by the array as a sequence of *transpositions*, where a transposition is simply a permutation of two elements, leaving all others fixed[137]. Performing the sequence of transpositions

$$(9,10),(8,9),(7,10),(6,8),(5,7),(4,9),(3,8),(2,7)$$

implements the permutation, matching the sequence of SWAPs in the circuit.

Having mapped the qubits, we compute the correct indices for the gate operations in the state preparation and verification circuit, as elucidated in the main text. After the entire state preparation routine, and before the noiseless syndrome extraction, we map the qubits back to their lexicographical ordering by applying the SWAP sequence in reverse (due to the omission of noise, the syndrome extraction is simulated without mapping and telegates for convenience).

[136] The DQC architecture here is represented as a list of indices, where consecutive numbers share the same core up to the respective capacity of the core.

[137] By combining transpositions, we can express any permutation.