LILLIPUT: A Lightweight Low-Latency Lookup-Table Decoder for Near-Term Quantum Error Correction

Poulami Das
poulami@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Aditya Locharla
adityalocharla@google.com
Google Quantum AI, Google Research
Mountain View, California, USA

Cody Jones
ncodyjones@google.com
Google Quantum AI, Google Research
Mountain View, California, USA

ABSTRACT

The error rates of quantum devices are orders of magnitude higher than what is needed to run most quantum applications. To close this gap, Quantum Error Correction (QEC) encodes logical qubits and distributes information using several physical qubits. By periodically executing a syndrome extraction circuit on the logical qubits, information about errors (called syndrome) is extracted while running programs. A decoder uses these syndromes to identify and correct errors in real time, which is necessary to prevent accumulation of errors. Unfortunately, software decoders are slow and hardware decoders are fast but less accurate. Thus, almost all QEC studies so far have relied on offline decoding.

To enable real-time decoding in near-term QEC, we propose LILLIPUT– a Lightweight Low Latency Look-Up Table decoder. LILLIPUT consists of two parts– First, it translates syndromes into error detection events that index into a Look-Up Table (LUT) whose entry provides the error information in real-time. Second, it programs the LUTs with error assignments for all possible error events by running a software decoder offline. LILLIPUT tolerates an error on any operation in the quantum hardware, including gates and measurements, and the number of tolerated errors grows with the size of the code. LILLIPUT utilizes less than 7% logic on off-the-shelf FPGAs enabling practical adoption, as FPGAs are already used to design the control and readout circuits in existing systems. LILLIPUT incurs a latency of a few nanoseconds and enables real-time decoding. We also propose Compressed LUTs (CLUTs) to reduce the memory required by LILLIPUT. By exploiting the fact that not all error events are equally likely and only storing data for the most probable error events, CLUTs reduce the memory needed by up-to 107x (from 148 MB to 1.38 MB) without degrading the accuracy.

CCS CONCEPTS

• **Hardware → Quantum technologies**; • **Computer systems organization → Quantum computing**.

KEYWORDS

Quantum error correction, Fault-tolerant quantum computing, Decoding, Lookup-Table decoder, Surface codes

ASPLOS '22, February 28-March 4, 2022, Lausanne, Switzerland

ACM ISBN 978-1-4503-9205-1/22/02…$15.00
https://doi.org/10.1145/3503222.3507707

ACM Reference Format:
Poulami Das, Aditya Locharla, and Cody Jones. 2022. LILLIPUT: A Lightweight Low-Latency Lookup-Table Decoder for Near-Term Quantum Error Correction. In *27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), February 28-March 4, 2022, Lausanne, Switzerland*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3503222.3507707

1 INTRODUCTION

Quantum computers promise substantial speedup over conventional machines for many important applications [48, 67, 75, 86]. Unfortunately, high error-rates of quantum devices (about 1% on existing hardware [7]) limit us from running these applications as they require much lower error-rates (below 10^{-10}) [34, 42, 44, 45]. To bridge this gap, quantum information must be protected by using Quantum Error Correction (QEC). QEC codes encode a logical qubit by distributing information over many physical qubits [5, 8, 31, 38, 74]. With increasing redundancy of the QEC code, the logical error-rate reduces exponentially if the physical error-rate is below a *threshold* [5]. Thus, by controlling the redundancy, QEC achieves the error-rate needed to run a particular quantum application.

QEC consist of both quantum and classical counterparts. On the quantum side, a logical qubit encodes quantum information in the combined state of multiple *data* qubits and uses *parity* qubits interspersed between them to gather information about errors, as shown in Figure 1(a). Each parity qubit periodically extracts the parity information of a subset of data qubits by executing a *syndrome extraction circuit* and projects the errors encountered by the data qubits into discrete Pauli errors. The process is repeated from the time qubits are initialized and until the data qubits are measured (called *logical measurement*). Each iteration of syndrome extraction is termed as a *QEC cycle* or *round* and the measurement outcome of the parity qubits is called a *syndrome*. On the classical side, a *decoder* uses the syndromes to detect errors and determine the most probable correction for the data qubits. The logical error-rate depends on the physical error-rate of the qubit devices as well as the performance of the decoder. High physical error rates make QEC codes ineffective [5]. Similarly, if a decoder is inaccurate or fails to correct errors in real-time, errors can accumulate. *Real-time decoding* refers to the detection and correction of errors dynamically as syndromes are received in each QEC cycle and before the arrival of the syndrome in the next cycle. Inaccurate decoding or failure to decode in real-time may cause logical failure during program execution. Thus, accurate real-time decoders are essential for QEC.

In recent years, several preliminary QEC experiments involving repetition codes and Bacon-Shor codes have been successfully demonstrated [6, 17, 18, 27, 49, 52, 66, 69, 71, 85, 87, 88, 90]. However, reaching low logical error rates requires implementation of more

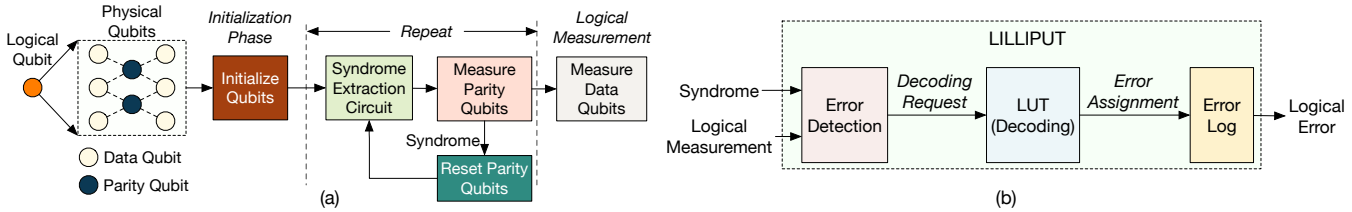

Figure 1: (a) In QEC, a logical qubit is encoded using a set of *data* and *parity* qubits. The parity qubits repeatedly extract the syndrome between qubit initialization and logical measurement. (b) LILLIPUT takes the syndrome and logical measurement outcomes as inputs, detects errors, obtains the error assignments for each data qubit from a Look-Up Table (LUT), and tracks an error log every QEC cycle. Finally, it computes the logical error by comparing the logical measurement and the error log.

efficient QEC codes such as surface codes [41], and recent studies [6, 9, 14, 50] have taken a step in this direction. Unfortunately, QEC studies so far have mainly resorted to offline decoding because most software decoders are slow [29]. Alternately, hardware decoders are faster but have poor accuracy. Moreover, hardware decoders need custom design [19] or superconducting devices [39, 84]. Custom decoding logic (such as an ASIC) is impractical for adoption in the near-term as it requires significant engineering effort, which may not be justifiable given the small number of quantum computers built today. Furthermore, some of the proposed designs may rely on technologies that are yet to mature for large-scale implementations (for example, superconducting designs). With improving device quality and size of quantum systems, QEC studies with small surface codes that span multiple QEC cycles and use real-time decoding represent the next significant milestone in quantum computing. Consequently, there is a growing demand and need for accurate, fast, and low-cost decoding solutions in near-term QEC. To address this challenge, we propose *LILLIPUT*– a Lightweight Low Latency Look-Up Table decoder for small surface codes.

Our proposed design, LILLIPUT, is an end-to-end system that directly interfaces with the qubit readout circuits, detects, and corrects errors in real-time, and computes the logical error. LILLIPUT performs three key steps in hardware, shown in Figure 1(b)– **(1)** translates the syndromes every QEC cycle into *error detection events*, **(2)** assigns errors to each data qubit from a Look-Up Table (LUT) and maintains an error log in real-time, and **(3)** computes the logical error by comparing the logical measurement outcomes with the most up-to-date error log. Additionally, LILLIPUT programs the LUT with the error assignments by running a software decoder offline for all possible error events.

For high accuracy, a decoder must correct (a) errors that accumulate on data qubits, (b) gate and (c) measurement errors in the syndrome extraction circuit, and (d) errors on data qubit measurements during the logical-measurement stage. LILLIPUT tackles all these errors by decoding multiple rounds of syndrome and by converting the logical measurement data into an appropriate syndrome in the last cycle. The accuracy and complexity of a decoder depends on the *decoder configuration*, a combination of the QEC code redundancy and the number of syndrome rounds used in decoding. As LILLIPUT is fully modular and reconfigurable, it can be implemented across a wide range of decoder configurations. LILLIPUT programs the LUTs offline using the software Minimum Weight Perfect Matching (MWPM) decoder [30, 32], widely used for its combination of high accuracy and polynomial time complexity. The MWPM decoder may produce multiple possible error assignments for a single error event. For greater accuracy, LILLIPUT considers the error model of the device (we use Google Sycamore [7]) and selects the most probable error assignment among all the possibilities. As LILLIPUT is reconfigurable, it can be adapted to other decoding algorithms, device error models, and QEC codes.

Instead of determining errors using hardware or software at run time, LILLIPUT transforms this step into a single memory access and therefore, incurs a deterministic low latency depending on the decoder configuration. LILLIPUT has low hardware complexity and fits on off-the-shelf FPGAs. As most existing quantum systems [1, 10] use FPGAs for delivery of control instructions to qubits and implementation of readout interface, LILLIPUT can be seamlessly integrated on these systems, and it attractive for practical adoption. Overall, LILLIPUT is accurate, fast, and lightweight which makes it an ideal candidate for real-time decoding in near-term QEC.

LILLIPUT incurs high memory overhead to store the LUTs and requires a memory external to the FPGAs for some decoder configurations. To address this challenge, we propose *Compressed Look-Up Tables (CLUTs)*. CLUTs exploit the fact that not all entries of a LUT are accessed with equal probability. This behavior arises from the nature of surface codes, where errors on data qubits only flip adjacent parity qubits. Infrequent errors flip a few parity qubits. Alternatively, more errors affect multiple locations resulting in longer error-chains. Such errors are more likely to flip the parity qubits back and forth lying on the error-chain, resulting in few bit flips on the surface code lattice overall. Consequently, the Hamming weight (number of ones) of the memory address accessed is typically low. CLUTs leverage this insight and only store entries corresponding to addresses of low Hamming weights that are most likely to be accessed. CLUTs determine the cut-off Hamming weight such that LILLIPUT encounters a decoder failure due to missing LUT entries with probability equal to or lower than the logical error rate and thus, has negligible impact on the overall accuracy.

The reconfigurability of LILLIPUT allows us to perform a design space exploration across various decoder configurations and understand the trade-off between the accuracy and complexity of a decoder. Our studies with small surface codes show that LILLIPUT achieves high accuracy. The decoding latency (time from the arrival of syndrome to error assignment) ranges between 29-42 ns for the decoder configurations studied in this paper. LILLIPUT requires up-to 7% logic utilization on off-the-shelf FPGAs [2–4] and thus, is lightweight. Lastly, CLUTs reduce the memory requirement of LILLIPUT by up-to 107x (from 148 MB to 1.3 MB).

Overall, this paper makes the following contributions:

(1) We propose *LILLIPUT*, a lightweight low latency lookup table decoder for small surface codes (requires <7% logic elements on FPGAs). It offers high accuracy as it can tolerate errors on both data qubits as well as in the syndrome extraction circuit; and requires a decoding latency within 42 ns.

(2) To the best of our knowledge, LILLIPUT is the first fully reconfigurable system-level decoding solution that can be seamlessly integrated with existing quantum systems.

(3) We propose *Compressed Look-Up Tables (CLUTs)* to reduce the memory overhead of LILLIPUT. By only storing the data for the most probable error events, CLUTs reduce the memory required by up-to 107x, without sacrificing the accuracy.

2 BACKGROUND AND MOTIVATION

2.1 Qubits and Quantum Error Correction

A quantum bit, or qubit, is the fundamental unit of information in a quantum computer. The state of a qubit can be represented as a superposition of its basis states $|0\rangle$ and $|1\rangle$, with complex valued amplitudes for these states [57]. Unfortunately, qubit devices retain information for only a short span of time (about few microseconds) and quantum operations have very high error rates (about 1%) [7]. These factors limit us from executing most quantum applications on existing hardware. To run programs without encountering errors, quantum information must be protected by using Quantum Error Correction (QEC) [5, 8, 31, 38, 74].

QEC codes encode a logical qubit using a set of data and parity qubits. The data qubits collectively store the quantum information, whereas the parity qubits periodically extract information about errors on the data qubits by executing a syndrome extraction circuit. Measuring the parity qubits every QEC cycle allows the QEC code to project any errors on the data qubits into a discrete set of Pauli errors. The bit-flip (X) error swaps the probability amplitudes of the basis states, whereas the phase-flip (Z) error introduces a relative phase of -1 between them. The Pauli Y error denotes simultaneous X and Z errors. The measurement outcomes of the parity qubits, called *syndrome*, is analyzed by a decoder to detect, and correct errors encountered by the data qubits. To perform computations, fault-tolerant quantum computers must perform error correction continuously while running an algorithm.

2.2 Surface Code

The surface code [26, 33, 41, 64] is widely considered to be the most promising QEC code as it can tolerate high thresholds [78] and requires only nearest-neighbor connectivity. It encodes a logical qubit into a 2-dimensional lattice of alternating data and parity qubits. The size and layout of the lattice depends on the code distance d which determines the code redundancy and the length of the shortest error chain ($\frac{d+1}{2}$) that cannot be corrected. An error on a data qubit is detected by the adjacent parity qubits by executing a syndrome extraction or stabilizer circuit, where each parity qubit measures a four-qubit operator called a *stabilizer*. X errors are detected by the Z stabilizers, whereas Z errors are detected by the X stabilizers [37]. For example, Figure 2(a) shows the layout of distance 3 regular surface code that can correct error chains of length 1. It consists of 13 data qubits (qubits A to M) and 12 parity qubits (qubits Z_0 to Z_6 and X_0 to X_6). An X error on data qubit D is detected by the Z stabilizers Z_0 and Z_2, whereas a Z error on this qubit is detected by the X stabilizers X_0 and X_1.

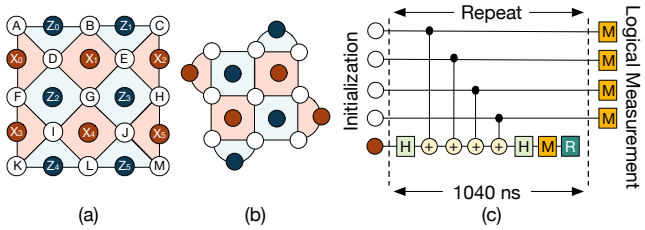

Figure 2: Distance 3 (a) regular and (b) rotated surface code. X errors are detected by the Z stabilizers (in blue) and Z errors are detected by the X stabilizers (in red). (c) Z stabilizer circuit

Figure 2(b) shows the layout of distance 3 *rotated surface code* that is obtained by rotating the lattice of a regular surface code by 45 degrees and removing some of the data and parity qubits. The rotated code requires fewer qubits and gate operations to extract a syndrome. Therefore, the rotated surface code is preferred over regular lattices for near-term QEC experiments, and we focus on rotated codes in this paper.

2.3 Error Decoding in Real-time

A decoder uses the output of stabilizer measurements, the syndrome, to determine a set of corrections that must be applied to the data qubits. By convention, the syndromes generated by the X stabilizers are called X syndromes and are decoded using an X decoder. Similarly, the syndromes generated by the Z stabilizers are called the Z syndromes and decoded using a Z decoder. Decoders must accurately identify errors in real-time to prevent accumulation of errors. The maximum latency that can be tolerated by a decoder is the latency of the syndrome extraction circuit, an example of which is shown in Figure 2(c). The circuit has a latency of about 1 μseconds, derived using specifications of existing device technology [6, 7]. Errors must be corrected within this time and designing accurate and fast decoders is an active area of research. Typically, software implementations are slow [29] and hence, more recently decoders have been proposed that uses custom hardware [19] and superconducting devices [39, 84] to enable fast decoding.

2.4 Motivation: Near-term QEC

QEC is essential to realize the potential of quantum computers in practice. As the device quality and system size continues to improve, there is increasing interest in studying QEC codes and real-time decoding. Several demonstrations of repetition codes, Bacon-Shor code have been successful [6, 17, 18, 27, 49, 52, 66, 69, 71, 85, 87, 88, 90]. While these experiments represent a significant milestone in QEC, unlike surface codes, these codes can only correct either phase-flip or bit-flip errors but not both. Also, QEC experiments so far have mainly relied on offline software-based error decoding. The largest surface code demonstrated so far is distance 2 [6, 9, 14, 50]. With improving device quality, quantum hardware may soon reach

the level of 1% error per data qubit per syndrome cycle, where QEC can function (involving surface codes of distance 3 and beyond). Thus, QEC experiments that demonstrate small surface codes using real-time decoding is a reasonable major milestone for quantum computing in the next few years.

2.5 Challenges in Real-time Decoding

Real-time decoding is necessary to prevent accumulation of errors on data qubits. If errors are not corrected within a QEC cycle before the arrival of the next syndrome, errors may accumulate resulting in a logical failure. The decoding complexity depends on the error events and software decoders may be too slow for real-time decoding [29]. They also incur significant communication overheads in transmitting the syndromes into software running on a CPU. This high latency of software decoders limits the use of general-purpose computing for online decoding and therefore, almost all QEC experiments that use software decoders have resorted to offline decoding. In the first instance of real-time decoding [70], software decoders have been used for color codes [77]. However, this study uses trapped-ion systems that can tolerate up to few milli-seconds of decoding latency, about 3 orders of magnitude higher than superconducting systems. The alternative is hardware decoders that are faster and promise real-time decoding. However, they have poor accuracy due to algorithmic and implementation limitations and rely on custom hardware [19] or specialized devices [39, 84]. Given the current state of superconducting device technology, it is not even feasible to implement the SFQ decoders [39, 84] in the near-term. The number of devices required to fabricate these decoders far exceeds the device densities of existing superconducting device technologies.

2.6 Goal: Decoding in Real-time for Near-term Quantum Error Correction

Ideally, we want a low-cost and accurate decoder that can be seamlessly integrated with existing quantum devices while enabling real-time decoding for near-term QEC experiments. To achieve this goal, we propose LILLIPUT– an accurate Lightweight Low Latency Look-Up Table decoder for small surface codes in this paper. We describe the evaluation methodology before discussing the design.

3 EVALUATION METHODOLOGY

In this section we describe the benchmarks, experimental setup, and the figure-of-merit used to evaluate our policies.

3.1 Surface Code Parameters

In this paper, we consider rotated surface codes of distance 3, 4, and 5. The details of the layouts are described in Table 1. The total number of physical qubits required ranges from 17 to 49. In the near-term, we expect systems with few hundreds of qubits which would be able to fit these layouts.

3.2 Monte Carlo Simulation Infrastructure

Figure 3 shows an overview of the Monte Carlo simulator used for our studies. The simulator generates a surface code lattice for a given code distance. Depending upon the noise model and the physical error rate p, the simulator injects errors onto the data qubits and measurement errors onto the parity qubits, producing a syndrome every cycle. The X and Z syndromes are then decoded independently, and an error log is maintained for both error types. To model a QEC experiment, the simulator repeats the process for multiple cycles and terminates when the maximum number of cycles is reached. The simulator maintains the internal state of the qubits throughout the experiment which is then used to compute the logical measurement outcome and the logical error. We call each such execution as a *trial*. For our evaluations, we use 1 million random trials. The simulator also generates the traces used to verify the proposed LILLIPUT micro-architecture.

Table 1: Parameters of Rotated Surface Codes

Code Distance	Data Qubits	X-ancilla Qubits	Z ancilla Qubits	Total Physical Qubits
3	9	4	4	17
4	16	8	7	31
5	25	12	12	49

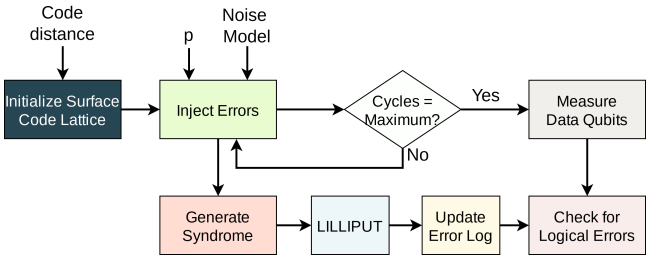

Figure 3: Monte Carlo simulation framework

3.3 Noise Model

We implement the *phenomenological noise model* [26], which inserts errors on both data qubits and on the measurement of parity qubits. This is a standard noise model used in QEC and has been used in several prior works. In this noise model, a data qubit encounters an error with probability p in each cycle. The type of error is chosen uniformly from Pauli X, Y, and Z errors. Additionally, each parity qubit encounters measurement error with probability p. For simplicity, we assume the probabilities for data qubit errors to be the same as measurement errors. This assumption is consistent with prior works in QEC. For our studies, we consider physical error rates ranging from $p = 10^{-3}$ to $p = 5 \times 10^{-2}$. We consider this to be a suitable range of error rates for quantum devices in the near term. Nonetheless, if the device quality improves further, our design can still support those quantum architectures.

3.4 Target Hardware Platforms

Our target is to implement the decoder on off-the-shelf FPGAs as existing quantum systems already use FPGAs for control and readout interface logic [1, 10, 11]. For our studies we use FPGA from the Intel Cyclone 10 LP [3], Arria V [2], and Stratix 10 [4] family as these are commercially available.

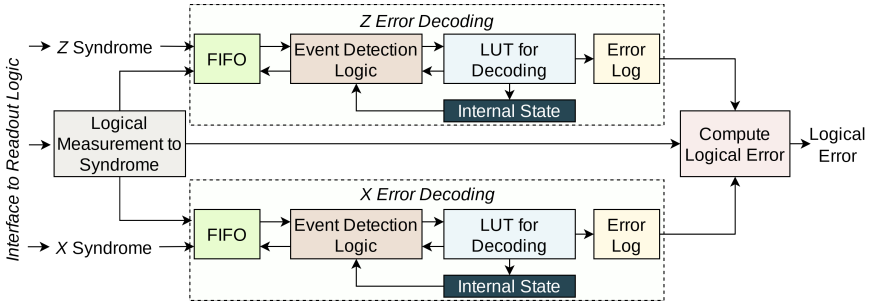

Figure 4: Overview of LILLIPUT

4 OVERVIEW OF LILLIPUT

The classical counterpart of QEC comprises of three key steps– **(1)** error detection from stabilizer measurements, **(2)** identification of errors and error assignment to data qubits every cycle, and **(3)** computation of logical error. Figure 4 shows the micro-architecture of LILLIPUT that accomplishes these steps. It communicates with the readout interface, translates the stabilizer measurement outcomes into error detection events, identifies errors, and computes the logical error. Note that X and Z errors are decoded independently. In the next subsections, we describe the design implementation.

4.1 Detection of Errors from the Stabilizer Measurements

In QEC, syndromes are generated every cycle by measuring the stabilizers. LILLIPUT generates *error detection events* by comparing the stabilizer measurement outcomes from two consecutive QEC cycles. Any change in the measurement outcome of the stabilizers between two cycles indicates an error. Alternately, no event is detected if the stabilizer measurement outcomes remain the same. For example, Figure 5 shows Z stabilizers measurements outcomes of a distance 3 surface code, where an error event is detected in cycles 1 and 3, whereas no error event is detected in cycle 2. This step is accomplished by the Event Detection Logic block shown in Figure 4. Detection events enables us to track errors in a given cycle and are used to identify the optimal correction.

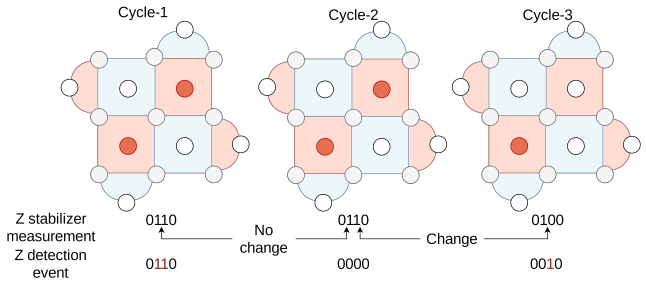

Figure 5: Steps involved in translating the stabilizer measurement outcomes (red denotes an error is identified) to error detection events in each QEC cycle. The bitstream for stabilizer measurements is specified from left to right (convention used in this paper) on the surface code lattice.

4.2 Error Identification as a Matching Problem

Error identification is the step in which a decoder assigns errors to each data qubit which is tracked throughout the QEC cycles. To perform this step, the detection events are represented on a graph, often termed as the *decoding graph*, where the nodes and edges represent the parity and data qubits respectively. The minimum weight perfect matching algorithm [32] uses the decoding graph and matches each detection node with another or to the surface code boundary such that the total weight of the matched edges is minimal. MWPM is widely considered to be the most accurate algorithm. A more recent method, the Union-Find algorithm [24, 25], uses a different approach to matching to generate the error assignments. It is faster than MWPM but has lower accuracy [40]. In LILLIPUT, instead of dynamically assigning errors in real-time, the matching step is accomplished using Look-Up Tables (LUTs), as shown in Figure 4, and the details are described in Section 5.

4.3 Handling Data and Measurement Errors

For greater accuracy, a decoder must handle both errors on the data qubits as well as errors in the syndrome extraction circuit. Overall, there are four key sources of errors: (a) errors on data qubits (b) errors in the gate operations during syndrome extraction (c) measurement errors on parity qubits during stabilizer measurements, and (d) measurement errors on data qubits during logical measurement. Next, we discuss how LILLIPUT handles each of these errors, where we use "space" and "time" directions to describe when detection events are generated in the same round or two consecutive cycles, respectively.

(a) An error on a data qubit is detected by its neighboring parity qubits, producing a ***space-like*** detection event on them in the same cycle. For example, Figure 6(a) shows a space-like error detection event in cycle 1.

(b) Two-qubit gate errors during syndrome extraction are detected across consecutive cycles. They produce ***space-time like*** detection events, as shown in cycles 1 and 3 in Figure 6(b).

(c) Measurement errors on parity qubits exhibit temporal behavior and translates into ***time-like*** detection events in two consecutive cycles, as shown in cycles 2 and 3 of Figure 6(b).

(d) Measurement errors on the data qubits are handled in the last boundary cycle by generating an extra syndrome of either X or Z type and is discussed in detail in the next subsection.

To tackle the first three sources of errors, LILLIPUT processes more than a single cycle of syndrome and performs matching on a 3-dimensional decoding graph spanning space and time (multiple cycles). For errors in logical measurement, LILLIPUT uses an extra syndrome as discussed in Section 4.4.

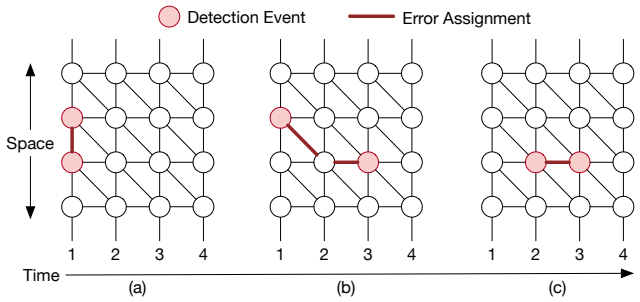

Figure 6: A distance 4 repetition code decoding graph, where nodes and edges denote parity and data qubits respectively. Note that this decoding graph is for the purpose of illustration only and our evaluations in this paper are based on surface codes. Examples of (a) space-like (b) space-time like and (c) time-like detection events.

LILLIPUT assumes the readout logic writes the measurement outcomes on a buffer which gets overwritten every cycle. This interface works on a much slower clock (e.g., with 1 μs latency) that depends on the duration of the syndrome extraction circuit. LILLIPUT polls for new syndromes and tracks the most recent history in a FIFO, as shown in Figure 4. The FIFO must store the history of the last m cycles, where m is the number of syndrome rounds that are simultaneously decoded. Since LILLIPUT decodes in real-time, storing only the last m rounds is sufficient. We discuss the impact of number of measurement rounds on the logical error rate in Section 6.

4.4 Handling Boundary Cycles

A QEC experiment has two time boundary cycles- one at the beginning and the other at the end. In QEC, cycles are also called *rounds*. We use the term cycles in this paper to avoid confusion with the number of syndrome rounds used for decoding. For the beginning time boundary cycle, the detection event is computed by comparing the first round of stabilizer measurements and the qubit initialization data. In the last time boundary cycle, the data qubits are measured. To tolerate measurement errors in data qubits, LILLIPUT translates the measurement outcomes of the data qubits into a detection event of either X or Z type by comparing with the stabilizer measurement data from the second-last cycle. Since a logical measurement in surface codes is implemented by performing a transversal measurement (measuring all the data qubits in either X or Z basis), the measurement basis of the data qubits determines the type of the stabilizer that can be constructed from the measurement outcomes. By default, LILLIPUT translates the data qubit measurement outcomes into a Z syndrome because we assume the data qubits are measured in the Z basis, as shown in Figure 4.

5 ERROR ASSIGNMENTS IN LILLIPUT

LILLIPUT determines the optimal error assignment using Look-Up Tables (LUTs). Every cycle, the history of detection events is used to index an LUT and the LUT entry assigns errors to the data qubits in the oldest cycle. The length of the history depends on the number of syndrome rounds used for decoding. LILLIPUT also maintains an *internal state* which is updated every cycle. The details of using LUTs in LILLIPUT and tracking the internal state is described next.

5.1 Streaming Mode of Operation

LILLIPUT operates in *streaming* mode and maintains an *error log* for each data qubit. The log is updated every cycle as errors are identified. LILLIPUT considers a fixed number of cycles at a time which is equal to the number of syndrome rounds used in decoding. We call this a *sliding window* because it slides forward as an experiment proceeds. For example, Figure 7 shows the sliding window for four consecutive decoding steps. LILLIPUT uses the detection events in the sliding window and the internal state to determine the LUT address every cycle.

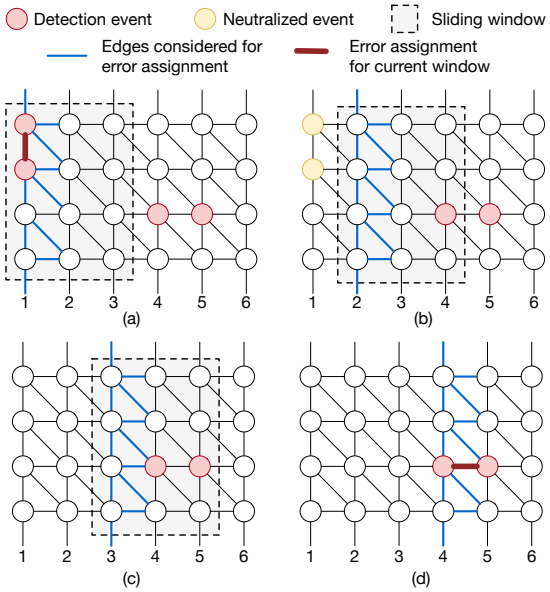

Figure 7: (a) The sliding window uses detection events from cycles 1, 2, and 3 and assigns error to the oldest layer i.e., cycle 1. Note that all detection events in the window are considered for matching but only the edges touching the oldest cycle are considered during error assignments, shown in blue. (b-d) The sliding window streams forward one cycle at a time.

5.2 Error Assignment

LILLIPUT determines the best correction by using all the detection events in each sliding window and assigns errors only to the oldest layer. For example, Figure 7 shows the subset of edges that the decoder uses to make the error assignment for the current sliding window. The optimal error assignment is stored in the LUTs and targets to *neutralize* the effect of all the errors in the oldest cycle

and decoding completes once the sliding window covers all the syndromes. The error assignments are made only to the oldest layer in each sliding window to prevent any *premature matching*. Premature matching may result in sub-optimal performance if syndromes in future cycles can change the assignment. Premature matching may also result if we use disjoint windows where the window proceeds forward by its full length. In that case, errors that span over two non-overlapping windows can cause inaccurate error assignments. As LILLIPUT uses a sliding window, it does not encounter this problem and therefore, does not suffer from sub-optimal performance. For the time boundary cycles, LILLIPUT constructs a full sliding window by padding zeros. Figure 8(c) shows a time boundary cycle where additional zeros are padded. Since the LUT is programmed such that, it assigns the most optimal error for a sliding window, it is guaranteed to never assign errors to the zero-padded regions.

5.3 Tracking the Internal State

LILLIPUT assigns errors only to the oldest layer. However, as it performs matching across all the events in a sliding window, it can neutralize some errors in the second oldest cycle if the detection event has a time-like or space-time like behavior. For example, Figure 7(d) shows a time-like detection event that requires neutralizing events in both cycles 4 (oldest cycle) and 5 (second oldest cycle). Consequently, matching within this window removes the detection event from the second oldest layer. Similarly, detection events may be added as well. For example, Figure 8(a) shows an example of decoding steps where detection events are added. To accommodate these scenarios, LILLIPUT maintains a record of the detection events added or removed in an internal state register. To obtain the LUT address to be accessed in a cycle, LILLIPUT concatenates the detection events in the current window and modifies the oldest detection event using the most recent internal state, as shown in Figure 8. Each LUT entry provides the error assignment for the oldest layer as well as the detection events that are added or removed in the current cycle (1 bit per parity qubit). This value is used to update the internal state register every cycle.

5.4 Programming the LUT

LILLIPUT programs the LUT entries offline by generating all possible detection events for a given code distance and size of the sliding window. We use a software Minimum Weight Perfect Matching (MWPM) decoder [30]. For some events, the MWPM decoder may provide more than one possible error assignment. In other words, a single detection event may have multiple matching possibilities on the decoding graph that result in minimal weight. For example, Figure 9 shows possible error assignments on a distance 4 surface code lattice for the same detection event. Here, the Z stabilizer at the center of the lattice indicates an error. The MWPM decoder can assign X errors to either data qubits (a) E and F or (b) G and H or (c) I and J. For high accuracy, we account for the error model of the quantum hardware (we consider Google Sycamore [6]) to select the most probable error assignment. LILLIPUT can also accommodate variability in device error rates [53, 80] by re-programming the LUTs. As device characteristics remain stable over short periods of time and mainly exhibit variability over extended periods (weeks or months) [23], the LUTs need not be re-programmed frequently.

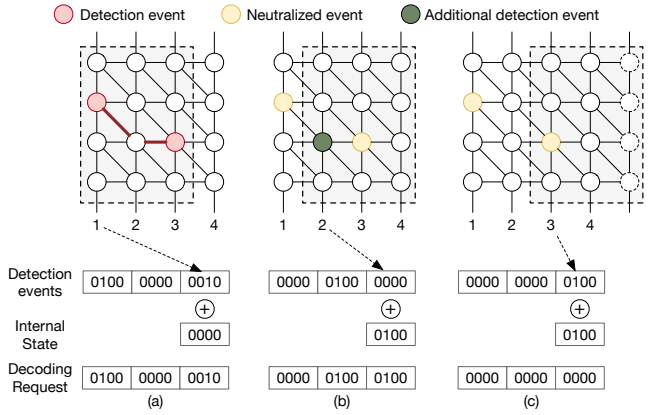

Figure 8: (a) LILLIPUT finds the optimal matching in the current window but adds a detection event in cycle 2 which is tracked in the internal state register. (b-c) The additional event is considered from the register. In the boundary cycle, zeros are padded. By convention, detection events of the oldest cycles occupy the least significant bits in our design. The bits for each cycle run bottom to top from most significant position to the least.

The reconfigurability allows LILLIPUT to be adapted to other QEC codes, decoding algorithms, and quantum systems.

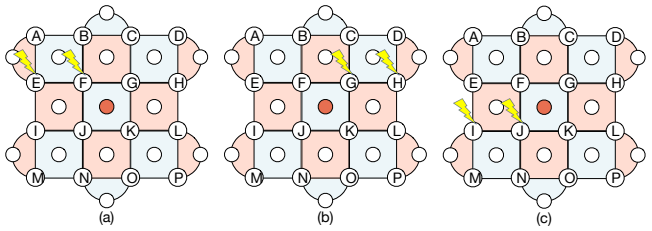

Figure 9: Valid possible error assignments for the detection event at the center of the distance 4 surface code lattice.

5.5 Decoding in presence of Non-Clifford Gates

We design LILLIPUT to operate on surface codes that use magic states for non-Clifford gates [31, 64]. A magic state is a resource state created using an unprotected physical non-Clifford gate that is 'injected' at the creation step of a surface-code logical qubit. A non-Clifford gate, such as T gate, is enacted by consuming this encoded magic state. This consumption process, which can include distillation of errors from magic states, is composed entirely of Clifford gates and measurements, as described in Section-XVI of [31].

An error on the creation of a magic state cannot be detected by any decoder, which is an unavoidable problem with state injection. To remedy this, an ensemble of noisy magic states is 'distilled' into one higher-fidelity magic state [15, 64]. Distillation is implemented by a circuit of Clifford operations, so this can be decoded using LILLIPUT (or another suitable decoder). Hence, LILLIPUT can support universal fault-tolerant logic in the surface code because all decoding occurs within Clifford gates.

5.6 Determination of Logical Error

Early demonstrations of QEC will perform a memory experiment, simply showing that error correction preserves the initial quantum state. The probability of logical error is determined by comparing the logical measurement outcome with the expected outcome for the state prepared. This logical measurement is computed using the error log. Which error log (X or Z) is used depends on the measurement basis of the logical measurement. The logical measurement bit is computed by using a reduction XOR operation on the bitwise XOR results of the error log and the logical measurement outcome.

6 FINAL EVALUATIONS

In this section, we discuss the accuracy, latency, and hardware complexity of our decoder.

6.1 Results for Accuracy

Figure 10 shows the logical error rate (LER) for distances (d) 3 and 4 respectively. By default, we assume 5 cycles in the experiments. The LER scales O(p^2) which is expected because distances 3 and 4 can correct at least one error, but they sometimes fail with two errors; quadratic scaling results from errors being independent in the model used for simulation. We also study the impact of the number of syndrome rounds on decoding accuracy. While $m = (d-1)$ rounds are needed to detect as many errors as the code is capable of correcting [78], we observe that LILLIPUT performance saturates at $m = 2$ syndrome rounds for $d = 4$. The LER reduces by 1.23x and 1.21x on average for distance 3 by going from 1 to 2 and 2 to 3 rounds respectively. Similarly, the LER reduces by 1.99x and 1.24x on average for distance 4 by going from 1 to 2 and 2 to 3 rounds respectively.

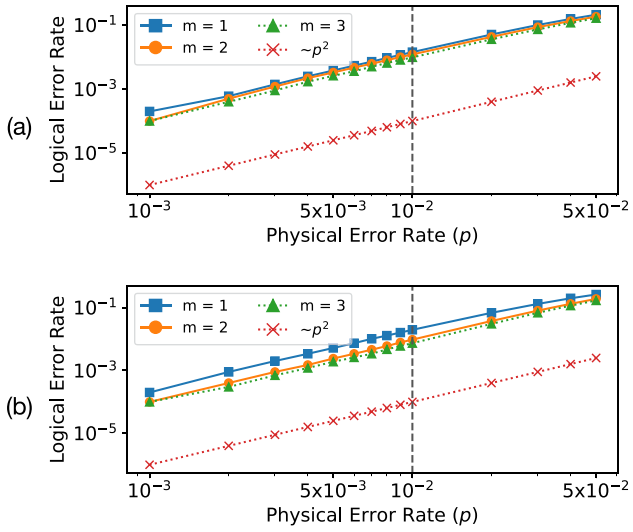

Figure 10: Accuracy of distance (a) 3 and (b) 4 surface codes for different number of syndrome measurement rounds (m)

Figure 11 shows the logical error rate for distance 5 surface codes using 1 and 2 rounds of syndrome measurements. We restrict the number of rounds to 2 to keep the LUT sizes tractable for our simulations. We observe that the LER scales O(p^2), which is expected as using just $m = 2$ rounds will lead to some configurations of two errors not being accurately corrected by the decoder. We also study the impact of number of cycles on the LER which is shown in Figure 12. This allows us to budget cycles in a QEC experiment based on the device error rates by accounting for the decoder performance.

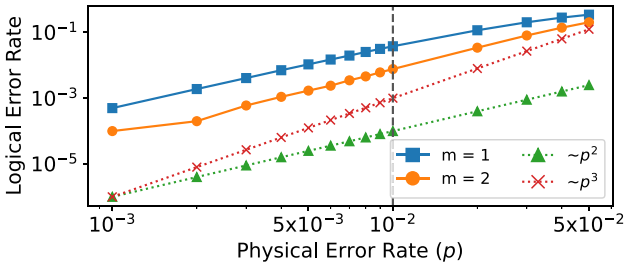

Figure 11: Accuracy of distance 5 surface codes for different number of syndrome measurement rounds (m)

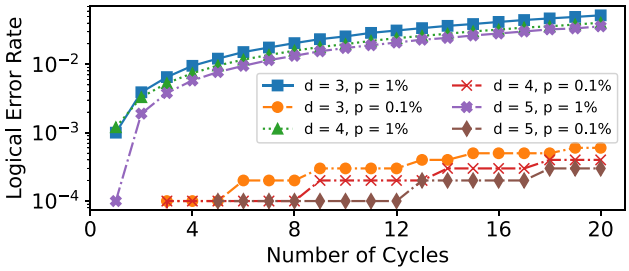

Figure 12: Impact of number of cycles on LER

6.2 Results for Hardware Complexity

The key component of LILLIPUT are the LUTs for both error types (X and Z). The size of each LUT depends on the *decoder configuration*. We denote a decoder configuration as $[d, m]$, where d and m are the code distance and number of syndrome rounds considered for decoding respectively. The address size is the length of the detection event which is equal to the syndrome length multiplied by the number of syndrome rounds. The length of each LUT entry is equal to the sum of the number of data (for error assignment) and parity qubits (for internal state). Table 2 shows the size of LUTs for different decoder configurations. Asymmetry in the code leads to X and Z syndrome lengths being different for distance 4, so the LUT sizes are different for X and Z syndromes. We use Cyclone 10 LP FPGAs for the decoder configurations $[d = 3, m = 2]$, $[d = 3, m = 3]$, and $[d = 4, m = 2]$ as it only supports up to 486 KB of embedded memory [3]. To support configurations with larger LUTs such as $[d = 4, m = 3]$, and $[d = 5, m = 2]$, we use Arria V FPGAs where the LUTs are accessed from an external SRAM (using QDR II) or SDRAM (using DDR2) [2]. Alternately, the LUTs can be compressed (discussed in Section 7) and does not require access to any external memory in that case [4].

Table 2: LUTs for different decoder configurations

Decoder Configuration	Address Size	Entry Size	LUT Size	Total Memory	Memory Type
$[d = 3, m = 2]$	8	13	416 B	832 B	Embedded
$[d = 3, m = 3]$	12	13	6.5 KB	13 KB	Embedded
$[d = 4, m = 2]$	14/ 16	23/ 24	46/ 192 KB	238 KB	Embedded
$[d = 4, m = 3]$	21/ 24	23/ 24	5.75/ 48 MB	53.75 MB	External
$[d = 5, m = 2]$	24	37	74 MB	148 MB	External

Table 3 mentions the FPGA utilization for different decoder configurations. The logic utilization for LILLIPUT is less than 7%, making it extremely lightweight and leaving enough room for other circuits such as readout interface logic and control logic for delivering instructions to the qubits. The LUTs consume up to 40% of the memory bits for designs that use embedded memory. We use different Cyclone devices for distance 3 and $[d = 4, m = 2]$ for higher performance. Configuring the distance-3 design on the latter FPGA reduces the maximum frequency to 245 MHz.

Table 3: FPGA utilization for different configurations

Decoder Configuration	FPGA Family	Total LEs/ ALMs	Total Registers	Utilization	
				Area	Memory
$[d = 3, m = 2]$	Cyclone 10	353	209	6%	1%
$[d = 3, m = 3]$	Cyclone 10	418	239	7%	21%
$[d = 4, m = 2]$	Cyclone 10	557	340	< 1%	40%
$[d = 4, m = 3]$	Arria-V	217	409	< 1%	–
$[d = 5, m = 2]$	Arria-V	246	486	< 1%	–

6.3 Results for Latency

LILLIPUT requires only a single memory access to the LUT and incurs fixed latency, irrespective of error detection event. LILLIPUT is fully pipelined and requires 7 cycles to determine the correction after a cycle of stabilizer measurements are received. The maximum clock-frequency depends on the decoder configuration and are listed in Table 4. To compute the latency of decoder configurations that rely on external memory access, we account for the most conservative off-chip memory access time (slowest clock frequency and maximum number of cycles) and add it to the latency of the rest of the logic. The decoding latency is up-to 24x lower than the target latency of 1μseconds in the worst-case (considering the largest decoder configuration studied in this paper). LILLIPUT also meets the 400 ns target latency considered in few prior works [19, 39, 84] and thus, will remain useful even if the duration of syndrome extraction decreases in future with improving device quality.

Table 4: Maximum Frequency (in MHz) and Latency (in ns)

Metric	Decoder Configurations				
	[d=3,m=2]	[d=3,m=3]	[d=4,m=2]	[d=4,m=3]	[d=5,m=2]
Frequency	250	240.7	209.8	244.4	232.9
Latency	28	29.1	33.4	40.8	42

The low logic utilization and re-usability of the LUTs allows LILLIPUT to support QEC experiments spanning more than one logical qubit. As there is sufficient slack between the decoding latency and the target latency for all decoder configurations studied in this paper, real-time decoding capability will not be impacted.

7 A CASE FOR COMPRESSED LUTS

The size of a lookup table increases exponentially in the code distance because there is an entry for every possible syndrome bit-string. This limits the scalability of LILLIPUT, and some of the decoder configurations studied in this paper rely on memory external to the FPGAs. Moreover, it is infeasible to scale the LUT to higher configurations, $[d = 5, m = 4]$ for example. To implement these configurations without external memory or scale LILLIPUT to other higher configurations, we propose the use of Compressed LUTs (CLUTs).

7.1 Not all Error Events are equally likely

Each LUT entry corresponds to an error event. However, not all error events are equally likely, and Compressed LUTs exploit this using the following insights:

(1) On surface codes, errors are detected by neighboring parity qubits. When error rates are low, fewer bit flips are observed in the detected events. Alternately, higher error-rates cause errors in multiple locations that result in longer chain of errors. However, it results in only few bit flips overall as the parity qubits inside the chain are likely to be zeros (can be thought of as flipped twice). Thus, the average Hamming weight of the accessed memory addresses is low. For example, the $[d = 3, m = 2]$ configuration requires LUTs with 8-bit address ranging from 0x00 to 0xFF. Figure 13 shows the probability distribution of the Hamming weight of addresses accessed over 1 million trials for different error rates and both X and Z LUTs of this decoder configuration. We observe that not all addresses are accessed with equal probability, and some are seldom or never accessed (address 0xFF for example). The trend persists even when the number of QEC cycles increases. This is expected because additional cycles lead to more errors and parity qubits flip back and forth between consecutive cycles. *Therefore, the memory required for the LUTs can be reduced by removing entries that are unlikely to be accessed.*

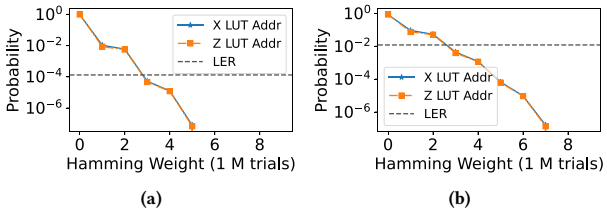

Figure 13: Probability distribution of the Hamming weight of memory addresses for (a) p = 0.1% and (b) p=1% for 5 cycles

(2) As for small distances, a decoder can assign only a limited number of errors, the LUT entries contain a large number of zeros. *Thus, the LUT entries themselves are compressible.*

To summarize, not all LUT entries are accessed with equal probability and therefore, the size of the LUTs can be reduced by eliminating entries that are unlikely to be accessed. Furthermore, the data entries corresponding to the most probable error events that must be stored can be compressed too.

7.2 Compressing the LUTs in Software

We investigate the scope of compression using the $[d = 3, m = 2]$ configuration. Figure 13 shows that the probability of accessing LUT addresses with 3 or more ones is equal to or lower than the logical error rate. Even if these entries are not stored, the decoder accuracy is unlikely to be affected as the events would result in a logical error with low probability.

To implement the CLUT, we split the address space into two groups– Segments A and B, as shown in Figure 14(a). These segments comprise of data frames (DFs)- a contiguous block of 16 and 10 entries respectively. A block of 16 consecutive LUT entries are assigned a 16-entry or 10-entry DF depending on the Hamming weight of the addresses of the block. Addresses with all zeros or a single one in the four most significant bits are assigned a 16-entry DF, whereas addresses with two ones in the four most significant bits are assigned a 10-entry DF. For example, as shown in Figure 14(a), 0x00 to 0x0F correspond to a 16-entry DF in Segment A, whereas 0xA0 to 0xAA are assigned a 10-entry DF in Segment B. Addresses 0xAB to 0xAF are not stored as their Hamming weight is more than 3. Although we want to store LUT entries for addresses of Hamming weights up-to 3 only, as memory addresses are not contiguous in terms of Hamming weight, this results in memory fragmentation. Instead, we use data frames that cause some addresses with higher Hamming weights to be stored as well (such as 0x1F), but this ensures a simpler addressing scheme. This reduces the number of entries by 1.83x from 256 to 140.

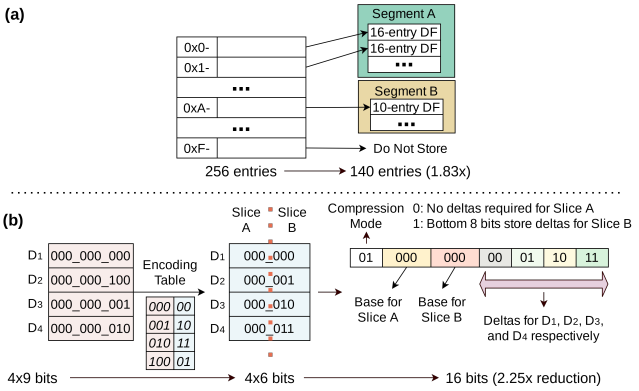

Figure 14: Steps (in software) involved in compressing LUTs.

Next, the DF entries are compressed by re-encoding using the least number of bits and Base-Delta-Immediate [62] compression. For example, 9-bit error assignments from four consecutive entries are packed into a 16-bit word, as shown Figure 14(b). First, each 9-bit data is encoded into 6-bit data using an encoding table. Next, four 6-bit entries are vertically sliced (Slice A and B) and packed into a 16-bit word. The 2-bit *compression mode* is needed because other memory regions require packing slice A and B differently. For example, when all slice B bits are 0s and slice A is stored using a 3-bit base and four 2-bit deltas, the mode is *10*. We investigated other compression schemes and found these to be most effective. Our studies show that for this configuration, compressing the internal state part of the DF entries needs a different scheme and the overheads of decompression exceed the benefits of compression as the data is only 4 bits. Finally, each CLUT reduces to 140 bytes, 3x lower than the full LUT (416 bytes). As this is done in software, it does not incur hardware overheads.

7.3 Accessing CLUTs and Decompression in Hardware

To service a decoding request, it is routed to the appropriate CLUT segment and DF. The CLUT entry is decompressed to obtain the error assignments and internal state, as shown in Figure 15. The hardware overhead to use CLUTs is the logic to obtain the segment address and perform decompression.

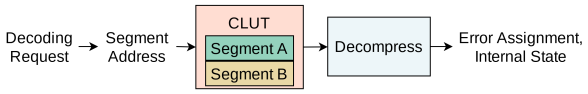

Figure 15: Steps (in hardware) involved in obtaining error assignment data from CLUTs.

7.4 Performance and Overheads of CLUTs

Figure 16(a) compares the logical error rate of the baseline design with respect to LILLIPUT using CLUTs. We observe that CLUTs offer similar performance as the baseline and does not degrade the LER. Figure 16(b) shows the decoder failure rate (due to missing LUT entries) and we observe that using CLUTs does not lead to events that increase the failure rate. A decoder failure here refers to failure to decode because the LUT address requested is not present in the CLUT.

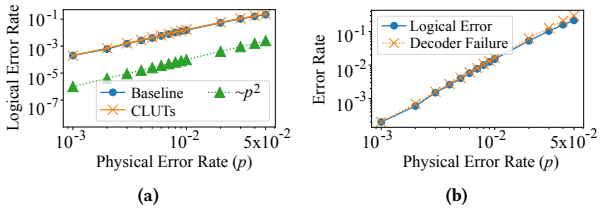

Figure 16: (a) Comparison of logical error rate for the baseline and LILLIPUT with CLUTs (b) Logical error rate and decoder failure rate for LILLIPUT with CLUTs

Table 5 shows the logic overhead (increases from 6% to 11%) and memory reduction (by 3x) in using CLUTs on Cyclone FPGAs. On Arria V, both designs require less than 1% logic utilization and thus, the overhead is acceptable.

Table 5: Logical Overhead for Implementing CLUTs

LILLIPUT (Baseline)			LILLIPUT with CLUT		
LEs	Registers	Memory	LEs	Registers	Memory
353 (6%)	209	832 Bytes	688 (11%)	261	280 Bytes

7.5 Scaling to Other Decoder Configurations

The LUTs for distance 4 and 5 surface codes can be reduced in a similar fashion. Figure 17 shows the probability distribution of the Hamming weight of the LUT addresses accessed for decoder configurations $[d = 4, m = 3]$ and $[d = 5, m = 2]$ respectively. We observe that storing only the LUT entries corresponding to Hamming weights 5 and below in the CLUTs are sufficient for both decoder configurations.

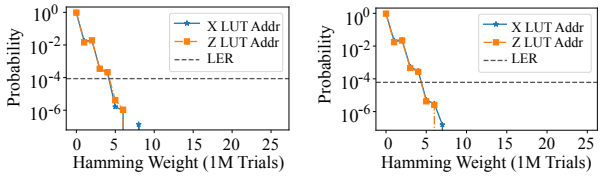

Figure 17: Probability distribution of Hamming weight of LUT accesses for (a) [d=4,m=3] and (b) [d=5,m=2]

Table 6 compares the size of CLUTs for distance 4 and 5 decoder configurations. We observe that using CLUTs can reduce the memory requirement by up-to 107x. The CLUTs fit within the embedded memory available on Arria II/V [2] and Stratix 10 devices [4] and does not require any external memory access. Note that this estimate does not consider compression of the DF entries in the CLUT (second step discussed in Section 7.2 as distances 4 and 5 require a different compression scheme and a detailed discussion is beyond the scope of this paper. To avoid the address translation for CLUT lookups, an alternative option is to use Cuckoo hashing [58] which incurs 2x memory overhead, but still fits the budget of these FPGA boards. Lastly, to support LILLIPUT for higher decoder configurations $[d = 5, m = 3]$ and $[d = 5, m = 4]$, the LUTs can be compressed and accessed from external memory.

Table 6: Memory Requirement for LILLIPUT with CLUTs

Decoder Configuration	Design	Memory (Baseline)		
		X	Z	Total
$[d = 4, m = 3]$	Baseline	5.75 MB	48 MB	53.75 MB
	W/ CLUT	245 KB	457 KB	702 KB (78.4x)
$[d = 5, m = 2]$	Baseline	74 MB	74 MB	148 MB
	W/ CLUT	704.6 KB	704.6 KB	1.38 MB (107x)

8 RELATED WORK

Noisy quantum devices limit us from running most applications as hardware error rates are orders of magnitude higher than what can be tolerated at the application-level. In the near-term, quantum computers with few hundreds of qubits [63] promise computational advantages for certain domain-specific applications [12, 13, 28, 51] with support from software error-mitigation [20–22, 35, 36, 46, 47, 53–56, 59–61, 61, 65, 72, 73, 76, 79, 81, 82]. However, fault-tolerant quantum computers can solve a broader class of problems [48, 75] and consequently, real-system QEC studies are gaining momentum [6, 68, 70] with improving device qualities and availability of quantum hardware prototypes.

In this section, we particularly discuss related work on decoders and compression and compare with LILLIPUT as appropriate.

Decoders for QEC: Developing accurate and fast decoders for QEC have been an area of research for several years. Recently, various hardware decoders [19, 39, 84] have been proposed for real-time decoding. However, these decoders have lower accuracy or rely on emerging technologies. In contrast, LILLIPUT is accurate, requires a constant latency, is reconfigurable, and can be seamlessly integrated with existing quantum systems, making it an ideal candidate in near future. Tomita et al. first investigated software LUT decoders using a fixed set of rules to determine the error assignments [83]. However, this design is only limited to distance 3 surface codes and 3 cycles. In contrast, LILLIPUT is an end-to-end solution that uses MWPM, accounts for device error models, and can be implemented over a wide number of configurations and QEC cycles. Most recently, Ryan-Anderson et al. used software decoders to achieve real-time decoding for color codes on trapped-ion systems [70]. However, this design requires access to general-purpose CPUs with the associated overhead in time for transferring data. Unfortunately, superconducting qubits may not tolerate such large latencies because unlike trapped ions, superconducting qubits can retain information for only a few microseconds. LILLIPUT on the other hand is more versatile and can be adapted to both device technologies. In general, there is no widely accepted standard decoding algorithm for color codes similar to MWPM for surface codes [16, 43].

Compression in conventional and quantum systems: Cache and memory compression in traditional architectures are mainly designed to optimize for capacity, bandwidth, and power. Their effectiveness varies depending on the sparsity of the data. Recently, Das et al. investigated syndrome compression for reducing the bandwidth needed for syndrome transmission in large quantum systems [19] by exploiting the sparsity in syndrome data. In contrast, compression of LUTs in LILLIPUT relies on the fact that not all error events are equally likely and error assignments data tend to be sparse. In another work [89], compression has been found to be effective in simulating quantum circuits for program verification.

9 CONCLUSION

Demonstration of small surface codes using real-time decoding represents a significant milestone in quantum computing. In this paper, we propose LILLIPUT- a Lightweight Low Latency Look-Up Table decoder that is accurate, fast, and can be seamlessly integrated with near-term quantum hardware. LILLIPUT interfaces with the readout logic, detects error events and corrects them as they appear in real-time. LILLIPUT uses LUTs to identify errors instead of running a decoding algorithm. The LUTs are programmed offline using a software decoder and accounts for the error model of the quantum device. In this paper, we use minimum weight perfect matching and error rates that are representative of a Google Sycamore device [7]. LILLIPUT utilizes less than 7% logic and requires up-to 148 MB of memory for distances up-to 5 on off-the-shelf FPGAs. To reduce the memory overhead of LILLIPUT, we propose Compressed LUTs (CLUTs). CLUTs exploit the fact that all error events are not equally likely and store information only for the most probable events. This reduces the memory required for the LUTs by up to 107x without deteriorating performance.

ACKNOWLEDGMENTS

This work was completed while Poulami Das was an intern in the Google Quantum AI group. We thank Kunal Arya, Michael Newman, Evan Jeffrey, Craig Gidney, and Austin Fowler for the technical discussions and feedback throughout the project. We thank Moin Qureshi for his inputs in improving the paper. We also thank the reviewers of ASPLOS-2022 for their comments and our shepherd Ali Javadi Abhari for his feedback in preparing the manuscript.

REFERENCES

[1] Accessed: August 03, 2021. Quantum Computing and IBM Q: An Introduction. https://www.e-shelter.de/sites/default/files/qubit_ibm_small.pdf.

[2] Accessed: July 22, 2021. External Memory Interface Handbook Volume 1: Intel FPGA Memory Solution Overview, Design Flow, and General Information. https://www.intel.com/content/www/us/en/programmable/documentation/hco1416493272601.html#hco1416492190668.

[3] Accessed: July 22, 2021. Intel Cyclone 10 LP Device Overview. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-10/c10lp-51001.pdf.

[4] Accessed: July 22, 2021. Intel Stratix 10 Embedded Memory User Guide Overview. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/ug-s10-memory.pdf.

[5] Dorit Aharonov and Michael Ben-Or. 1999. Fault-tolerant quantum computation with constant error rate. *arXiv preprint quant-ph/9906129* (1999).

[6] Google Quantum AI. 2021. Exponential suppression of bit or phase errors with cyclic error correction. *Nature* 595, 7867 (2021), 383.

[7] Google Quantum AI. Accessed: June 19, 2021. Quantum Computer Datasheet. https://quantumai.google/hardware/datasheet/weber.pdf.

[8] Panos Aliferis, Daniel Gottesman, and John Preskill. 2005. Quantum accuracy threshold for concatenated distance-3 codes. *arXiv quant-ph/0504218* (2005).

[9] Christian Kraglund Andersen, Ants Remm, Stefania Lazar, Sebastian Krinner, Nathan Lacroix, Graham J Norris, Mihai Gabureac, Christopher Eichler, and Andreas Wallraff. 2020. Repeated quantum error detection in a surface code. *Nature Physics* 16, 8 (2020), 875–880.

[10] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[11] Joseph C Bardin, Evan Jeffrey, Erik Lucero, Trent Huang, Ofer Naaman, Rami Barends, Ted White, Marissa Giustina, Daniel Sank, Pedram Roushan, et al. 2019. 29.1 A 28nm Bulk-CMOS 4-to-8GHz¡ 2mW Cryogenic Pulse Modulator for Scalable Quantum Computing. In *ISSCC*. IEEE, 456–458.

[12] Joao Basso, Edward Farhi, Kunal Marwaha, Benjamin Villalonga, and Leo Zhou. 2021. The Quantum Approximate Optimization Algorithm at High Depth for MaxCut on Large-Girth Regular Graphs and the Sherrington-Kirkpatrick Model. *arXiv preprint arXiv:2110.14206* (2021).

[13] Bela Bauer, Sergey Bravyi, Mario Motta, and Garnet Kin-Lic Chan. 2020. Quantum algorithms for quantum chemistry and quantum materials science. *Chemical Reviews* 120, 22 (2020), 12685–12717.

[14] BA Bell, DA Herrera-Martí, MS Tame, Damian Markham, WJ Wadsworth, and JG Rarity. 2014. Experimental demonstration of a graph state quantum error-correction code. *Nature communications* 5, 1 (2014), 1–10.

[15] Sergey Bravyi and Alexei Kitaev. 2005. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A* 71 (Feb 2005), 022316. Issue 2. https://doi.org/10.1103/PhysRevA.71.022316

[16] Christopher Chamberland, Aleksander Kubica, Theodore J Yoder, and Guanyu Zhu. 2020. Triangular color codes on trivalent graphs with flag qubits. *New Journal of Physics* 22, 2 (2020), 023019.

[17] David G Cory, MD Price, W Maas, Emanuel Knill, Raymond Laflamme, Wojciech H Zurek, Timothy F Havel, and Shyamal S Somaroo. 1998. Experimental quantum error correction. *PRL* 81, 10 (1998), 2152.

[18] Julia Cramer, Norbert Kalb, M Adriaan Rol, Bas Hensen, Machiel S Blok, Matthew Markham, Daniel J Twitchen, Ronald Hanson, and Tim H Taminiau. 2016. Repeated quantum error correction on a continuously encoded qubit by real-time feedback. *Nature communications* 7, 1 (2016), 1–7.

[19] Poulami Das, Christopher A Pattison, Srilatha Manne, Douglas Carmean, Krysta Svore, Moinuddin Qureshi, and Nicolas Delfosse. 2020. A scalable decoder micro-architecture for fault-tolerant quantum computing. *arXiv:2001.06598* (2020).

[20] Poulami Das, Swamit Tannu, Siddharth Dangwal, and Moinuddin Qureshi. 2021. ADAPT: Mitigating Idling Errors in Qubits via Adaptive Dynamical Decoupling. In *MICRO-54*.

[21] Poulami Das, Swamit Tannu, and Moinuddin Qureshi. 2021. JigSaw: Boosting Fidelity of NISQ Programs via Measurement Subsetting. In *MICRO-54*.

[22] Poulami Das, Swamit S Tannu, Prashant J Nair, and Moinuddin Qureshi. 2019. A Case for Multi-Programming Quantum Computers. In *MICRO*. ACM, 291–303.

[23] Samudra Dasgupta and Travis S Humble. 2021. Stability of noisy quantum computing devices. *arXiv preprint arXiv:2105.09472* (2021).

[24] Nicolas Delfosse and Naomi H Nickerson. 2017. Almost-linear time decoding algorithm for topological codes. *arXiv preprint arXiv:1709.06218* (2017).

[25] Nicolas Delfosse and Gilles Zémor. 2017. Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel. *arXiv:1703.01517* (2017).

[26] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. 2002. Topological quantum memory. *J. Math. Phys.* 43, 9 (2002), 4452–4505.

[27] Laird Egan, Dripto M Debroy, Crystal Noel, Andrew Risinger, Daiwei Zhu, Debopriyo Biswas, Michael Newman, Muyuan Li, Kenneth R Brown, Marko Cetina, et al. 2020. Fault-tolerant operation of a quantum error-correction code. *arXiv preprint arXiv:2009.11482* (2020).

[28] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).

[29] Austin Fowler. 2017. Towards sufficiently fast quantum error correction. (2017). https://qec2017.gatech.edu Conference QEC 2017.

[30] Austin G Fowler. 2013. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $O(1)$ parallel time. *arXiv preprint arXiv:1307.1740* (2013).

[31] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A* 86 (Sep 2012), 032324. Issue 3. https://doi.org/10.1103/PhysRevA.86.032324

[32] Austin G Fowler, Adam C Whiteside, and Lloyd CL Hollenberg. 2012. Towards practical classical processing for the surface code. *PRL* 108, 18 (2012), 180501.

[33] Austin G Fowler, Adam C Whiteside, and Lloyd CL Hollenberg. 2012. Towards practical classical processing for the surface code. *PRL* 108, 18 (2012), 180501.

[34] Craig Gidney and Martin Ekerå. 2021. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* 5 (2021), 433.

[35] Pranav Gokhale, Yongshan Ding, Thomas Propson, Christopher Winkler, Nelson Leung, Yunong Shi, David I Schuster, Henry Hoffmann, and Frederic T Chong. 2019. Partial Compilation of Variational Algorithms for Noisy Intermediate-Scale Quantum Machines. In *MICRO*. ACM, 266–278.

[36] Pranav Gokhale, Ali Javadi-Abhari, Nathan Earnest, Yunong Shi, and Frederic T Chong. 2020. Optimized Quantum Compilation for Near-Term Algorithms with OpenPulse. *arXiv:2004.11205* (2020).

[37] Daniel Gottesman. 1998. Theory of fault-tolerant quantum computation. *Phys. Rev. A* 57 (Jan 1998), 127–137. Issue 1. https://doi.org/10.1103/PhysRevA.57.127

[38] Daniel Gottesman. 2010. An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum information science and its contributions to mathematics, Proc. of Symposia in Applied Mathematics*, Vol. 68.

[39] Adam Holmes, Mohammad Reza Jokar, Ghasem Pasandi, Yongshan Ding, Massoud Pedram, and Frederic T Chong. 2020. NISQ+: Boosting quantum computing power by approximating quantum error correction. *arXiv preprint arXiv:2004.04794* (2020).

[40] Shilin Huang, Michael Newman, and Kenneth R. Brown. 2020. Fault-tolerant weighted union-find decoding on the toric code. *Phys. Rev. A* 102 (Jul 2020), 012419. Issue 1. https://doi.org/10.1103/PhysRevA.102.012419

[41] A Yu Kitaev. 2003. Fault-tolerant quantum computation by anyons. *Annals of Physics* 303, 1 (2003), 2–30.

[42] Ian D Kivlichan, Craig Gidney, Dominic W Berry, Nathan Wiebe, Jarrod McClean, Wei Sun, Zhang Jiang, Nicholas Rubin, Austin Fowler, Alán Aspuru-Guzik, et al. 2020. Improved fault-tolerant quantum simulation of condensed-phase correlated electrons via Trotterization. *Quantum* 4 (2020), 296.

[43] Andrew J Landahl, Jonas T Anderson, and Patrick R Rice. 2011. Fault-tolerant quantum computing with color codes. *arXiv preprint arXiv:1108.5738* (2011).

[44] Joonho Lee, Dominic W Berry, Craig Gidney, William J Huggins, Jarrod R McClean, Nathan Wiebe, and Ryan Babbush. 2020. Even more efficient quantum computations of chemistry through tensor hypercontraction. *arXiv preprint arXiv:2011.03494* (2020).

[45] Jessica Lemieux, Guillaume Duclos-Cianci, David Sénéchal, and David Poulin. 2021. Resource estimate for quantum many-body ground-state preparation on a quantum computer. *Phys. Rev. A* 103, 5 (2021), 052408.

[46] Gushu Li, Yufei Ding, and Yuan Xie. 2018. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. *arXiv preprint arXiv:1809.02573* (2018).

[47] Gushu Li, Anbang Wu, Yunong Shi, Ali Javadi-Abhari, Yufei Ding, and Yuan Xie. 2021. On the Co-Design of Quantum Software and Hardware. In *Nanoscale Computing and Communication Conference*. 1–7.

[48] Seth Lloyd. 1996. Universal quantum simulators. *Science* (1996), 1073–1078.

[49] Yi-Han Luo, Ming-Cheng Chen, Manuel Erhard, Han-Sen Zhong, Dian Wu, Hao-Yang Tang, Qi Zhao, Xi-Lin Wang, Keisuke Fujii, Li Li, et al. 2020. Quantum teleportation of physical qubits into logical code-spaces. *arXiv:2009.06242* (2020).

[50] JF Marques, BM Varbanov, MS Moreira, Hany Ali, Nandini Muthusubramanian, Christos Zachariadis, Francesco Battistel, Marc Beekman, Nadia Haider, Wouter Vlothuizen, et al. 2021. Logical-qubit operations in an error-detecting surface

code. *arXiv preprint arXiv:2102.13071* (2021).
[51] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. 2016. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics* 18, 2 (2016), 023023.
[52] Osama Moussa, Jonathan Baugh, Colm A Ryan, and Raymond Laflamme. 2011. Demonstration of sufficient control for two rounds of quantum error correction in a solid state ensemble quantum information processor. *PRL* 107, 16 (2011).
[53] Prakash Murali, Jonathan M Baker, Ali Javadi Abhari, Frederic T Chong, and Margaret Martonosi. 2019. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. *ASPLOS* (2019).
[54] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. 2019. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *ISCA*. IEEE, 527–540.
[55] Prakash Murali, Norbert M Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. 2020. Architecting Noisy Intermediate-Scale Quantum Computers: A Real-System Study. *IEEE Micro* 40 (2020).
[56] Prakash Murali, David C McKay, Margaret Martonosi, and Ali Javadi-Abhari. 2020. Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In *ASPLOS*. 1001–1016.
[57] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.
[58] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122–144.
[59] Tirthak Patel and Devesh Tiwari. 2020. Veritas: accurately estimating the correct output on noisy intermediate-scale quantum computers. In *SC20*. IEEE.
[60] Tirthak Patel and Devesh Tiwari. 2021. Qraft: reverse your Quantum circuit and know the correct program output. In *ASPLOS*.
[61] Tirthak Patel, Ed Younis, Costin Iancu, Wibe de Jong, and Devesh Tiwari. 2021. Robust and Resource-Efficient Quantum Circuit Approximation. *arXiv preprint arXiv:2108.12714* (2021).
[62] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. 2012. Base-delta-immediate compression: practical data compression for on-chip caches. In *PACT*. ACM, 377–388.
[63] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (2018), 79.
[64] Robert Raussendorf and Jim Harrington. 2007. Fault-Tolerant Quantum Computation with High Threshold in Two Dimensions. *Phys. Rev. Lett.* 98 (May 2007), 190504. Issue 19. https://doi.org/10.1103/PhysRevLett.98.190504
[65] Gokul Subramanian Ravi, Kaitlin N Smith, Pranav Gokhale, Andrea Mari, Nathan Earnest, Ali Javadi-Abhari, and Frederic T Chong. 2021. VAQEM: A Variational Approach to Quantum Error Mitigation. *arXiv preprint arXiv:2112.05821* (2021).
[66] Matthew D Reed, Leonardo DiCarlo, Simon E Nigg, Luyan Sun, Luigi Frunzio, Steven M Girvin, and Robert J Schoelkopf. 2012. Realization of three-qubit quantum error correction with superconducting circuits. *Nature* 482, 7385 (2012).
[67] Markus Reiher, Nathan Wiebe, Krysta M Svore, Dave Wecker, and Matthias Troyer. 2017. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences* 114, 29 (2017), 7555–7560.
[68] Ants Remm, Christian Kraglund Andersen, Stefania Lazar, Sebastian Krinner, Nathan Lacroix, Christoph Hellings, Agustin Di Paolo, Francois Swiadek, Graham Norris, Johannes Hermann, et al. 2021. Quantum Error Correction Using a Distance Three Surface Code with Superconducting Qubits. *APS* (2021).
[69] Diego Riste, Stefano Poletto, M-Z Huang, Alessandro Bruno, Visa Vesterinen, O-P Saira, and Leonardo DiCarlo. 2015. Detecting bit-flip errors in a logical qubit using stabilizer measurements. *Nature communications* 6, 1 (2015), 1–6.
[70] C Ryan-Anderson, JG Bohnet, K Lee, D Gresh, A Hankin, JP Gaebler, D Francois, A Chernoguzov, D Lucchetti, NC Brown, et al. 2021. Realization of real-time fault-tolerant quantum error correction. *arXiv preprint arXiv:2107.07505* (2021).
[71] Philipp Schindler, Julio T Barreiro, Thomas Monz, Volckmar Nebendahl, Daniel Nigg, Michael Chwalla, Markus Hennrich, and Rainer Blatt. 2011. Experimental repetitive quantum error correction. *Science* 332, 6033 (2011), 1059–1061.
[72] Yunong Shi, Pranav Gokhale, Prakash Murali, Jonathan M Baker, Casey Duckering, Yongshan Ding, Natalie C Brown, Christopher Chamberland, Ali Javadi-Abhari, Andrew W Cross, et al. 2020. Resource-Efficient Quantum Computing by Breaking Abstractions. *Proc. IEEE* 108, 8 (2020), 1353–1370.
[73] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I Schuster, Henry Hoffmann, and Frederic T Chong. 2019. Optimized compilation of aggregated instructions for realistic quantum computers. In *ASPLOS*.
[74] Peter W Shor. 1996. Fault-tolerant quantum computation. In *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE, 56–65.
[75] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computers. *SIAM review* 41, 2 (1999), 303–332.
[76] Kaitlin N Smith, Gokul Subramanian Ravi, Prakash Murali, Jonathan M Baker, Nathan Earnest, Ali Javadi-Abhari, and Frederic T Chong. 2021. Error Mitigation in Quantum Computers through Instruction Scheduling. *arXiv preprint arXiv:2105.01760* (2021).
[77] Andrew M Steane. 1996. Error correcting codes in quantum theory. *PRL* 77, 5 (1996), 793.
[78] Ashley M. Stephens. 2014. Fault-tolerant thresholds for quantum error correction with the surface code. *Phys. Rev. A* 89 (Feb 2014), 022321. Issue 2.
[79] Swamit S Tannu and Moinuddin Qureshi. 2019. Ensemble of Diverse Mappings: Improving Reliability of Quantum Computers by Orchestrating Dissimilar Mistakes. In *MICRO*. ACM, 253–265.
[80] Swamit S Tannu and Moinuddin K Qureshi. 2018. A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. *arXiv:1805.10224* (2018).
[81] Swamit S Tannu and Moinuddin K Qureshi. 2019. Mitigating Measurement Errors in Quantum Computers by Exploiting State-Dependent Bias. In *MICRO*. ACM.
[82] Swamit S Tannu and Moinuddin K Qureshi. 2019. Not All Qubits are Created Equal: A Case for Variability-Aware Policies for NISQ-era Quantum Computers. In *ASPLOS*. ACM, 987–999.
[83] Yu Tomita and Krysta M Svore. 2014. Low-distance surface codes under realistic quantum noise. *Phys. Rev. A* 90, 6 (2014), 062320.
[84] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. 2021. QECOOL: On-Line Quantum Error Correction with a Superconducting Decoder for Surface Code. *arXiv preprint arXiv:2103.14209* (2021).
[85] Gerald Waldherr, Y Wang, S Zaiser, M Jamali, T Schulte-Herbrüggen, H Abe, T Ohshima, J Isoya, JF Du, P Neumann, et al. 2014. Quantum error correction in a solid-state hybrid spin register. *Nature* 506, 7487 (2014), 204–207.
[86] Dave Wecker, Matthew B Hastings, Nathan Wiebe, Bryan K Clark, Chetan Nayak, and Matthias Troyer. 2015. Solving strongly correlated electron models on a quantum computer. *Phys. Rev. A* 92, 6 (2015), 062318.
[87] James R Wootton. 2020. Benchmarking near-term devices with quantum error correction. *Quantum Science and Technology* 5, 4 (2020), 044004.
[88] James R Wootton and Daniel Loss. 2018. Repetition code of 15 qubits. *Phys. Rev. A* 97, 5 (2018), 052313.
[89] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T Chong. 2019. Full-state quantum circuit simulation by using data compression. In *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–24.
[90] Jingfu Zhang, Dorian Gangloff, Osama Moussa, and Raymond Laflamme. 2011. Experimental quantum error correction with high fidelity. *Phys. Rev. A* 84 (2011).