SWISS NANOSCIENCE INSTITUTE
EINE INITIATIVE DER UNIVERSITÄT BASEL UND DES KANTONS AARGAU

UNI BASEL

Quantum Error Correction Benchmark and Diagnostics

Master's Thesis

Milan Liepelt
m.liepelt@stud.unibas.ch

Quantum Computational Science, IBM Research GmbH
Swiss Nanoscience Institute, University of Basel

Supervisors:
Dr. James Wootton, IBM Research GmbH
Prof. Dr. Daniel Loss, University of Basel

June 1, 2023

First and foremost I want to thank my supervisors Dr. James Wotton and Prof. Daniel Loss for the opportunity, the support, and the feedback for my master's thesis. Furthermore, I want to thank IBM Quantum and the NCCR Spin for hosting my master's thesis and granting me access to the IBM quantum computers and other computer infrastructure. Furthermore, I want to thank Dr. Bence Hetenyi, Dr. Daniel Egger, Conrad Haupt, Ian Hesner, Mario Chizini and Tommaso Peduzzi for their valuable discussions and input. Thanks goes also to the Swiss Nanoscience Institute and the University of Basel to make this thesis possible and for administrative support. My deepest thanks go to my partner Adrienne Müller for her invaluable emotional, logistical, accommodation support during this time and her feedback to the manuscript.

Abstract

With the improvements in the quality and size of quantum computers, the point where quantum error correction (QEC) can be the key-enabling technology for specific real-world applications comes closer. Quantum computing platforms differ in their physical implementation and architectures and have therefore different strengths and weaknesses, which can make QEC more or less effective. To assess the potential that can be reached with QEC reliably, we present a method for quantum error correction diagnostics and benchmarking. The QEC benchmark allows to assess and compare how well a quantum computer retains useful quantum information over time and how well this number scales by adding more qubits to the QEC system. The developed method is based on the anisotropic repetition code (ARC). This codes' unique property is the combination of the simplicity of a standard repetition code, while encoding neighboring data qubits in different bases, which is a special property of more advanced QEC codes like the Surface code, and might be an additional source of noise. This code allows further data to be gathered for the so-called micro diagnostics/benchmarks to assess a quantum computer down to single qubit properties. Different diagnostic methods based on syndrome data were developed to detect erroneous qubits and leakage during QEC runtime. Such information can be useful for quantum circuit schedulers to optimize which qubits to run an algorithm best, and QEC decoders can also greatly benefit from this information to improve accuracy. Finally, a protocol is proposed under which the ARC might qualify to be taken up in application-oriented quantum computing benchmarking suites.

Contents

Chapter 1

Introduction

1.1 Benchmarks in Quantum Computing

With the availability of different commercial quantum computers in the cloud, it becomes more important to have a possibility to compare their performance and capabilities for different applications. Many quantum computing suppliers offer on their websites and platforms information about hardware properties, quality, and error rates like the T1, and T2 thermal relaxation times, one and two-qubit gate error rates, state preparation and readout error rates, measured with randomized benchmarking or other characterization protocols [1–3]. For a quantum computing user who is not specialized in quantum hardware systems, it might be challenging to decide on which platform an algorithm of a certain class runs best, based on such system benchmarks [4].

A first approach to compare the capabilities of a quantum computer is the quantum volume Q_V [5]. This takes into account the number of qubits, the ability to run deep and wide circuits, the error rates, the connectivity of the qubits and the compiler efficiency and outputs a single number, known as the quantum volume Q_V. Q_V is approximately $Q_V = 2^d$ where d is the number of qubits and the depth of the circuit that can produce a heavy output of randomized circuits successfully [6]. To date (May 2023), the quantum computer with the highest known quantum volume of 65536 is the trapped ion-based Quantinuum H2 which means the random model circuit could be run with d=16 qubits. IBM's current high score is a $Q_V = 512$ on their machine ibm_prague which means the model circuit could be run successfully with 9 qubits [7]. Unfortunately, this quantum computer was not available for this study, so we will rather work with quantum computers with quantum volumes ranging from 32 to 128, or the Q_V is not yet defined or published in the case of ibm_seattle with 433 qubits as of mid-May 2023. The quantum volume itself is criticized to become something like a consumer index and companies start to optimize their systems to just perform well on the quantum volume score, but that does not make them necessarily better at executing real and useful tasks [8]. Other criticisms are, that the quantum volume only measures the square-shaped circuits (same number for depth and width), but there are important quantum algorithms like Quantum Fourier Transform (QFT), or Grover's search that are shallow and wide, or narrow and deep.

Fortunately, a new class of benchmarks emerged. 'Application-Oriented Performance Benchmarks for Quantum Computing', developed by the Quantum Economic Development Consortium (QED-C) takes into account a variety of applications and codes [9] and assess how well the quantum computers perform on those. Another benchmark, the 'Application-Motivated, Holistic Benchmarking of a Full Quantum Computing Stack'

proposes specific benchmarking codes that consider the different shapes of the quantum algorithms [10].

SupermarQ [11] is an additional benchmarking suite, which is based on various quantum algorithms and codes and is built around design principles like scalability to cover and remain relevant up to quantum supremacy capable systems, which means that there are no exponentially classical processes involved to simulate the output of an ideal result, to compare the quantum computers' results against. Other design principles are meaningful and diverse which means to use different and useful quantum algorithms. Full-system evaluation assesses the whole quantum computing system including the circuit compiler. Adaptivity stands for the fact that the benchmarking suite can be adapted if necessary to include new algorithms or adapt to new developments. It even covers quantum error correction algorithms, in the form of a bit flip and a phase flip repetition code up to a distance of 5 and with 5 repetitions to cover this very relevant topic towards fault-tolerant quantum computing. The quantum error correction codes are the only ones covering mid-circuit measurements but perform very poorly in terms of its benchmarking score compared to other algorithms. The authors point out that mid-circuit measurements and reset operations could be responsible for the bad performance, since those operations take very long compared to other gates, and the limited T1 and T2 times cause that the quantum information is lost relatively quickly, due to the prolonged runtime of each circuit [11] .

Quantum error correction (QEC) is an approach to mitigate the information loss in quantum computers due to noise and errors, other unwanted interactions of the qubits with the environment or with each other, or errors caused by qubits that leave their computational subspace space and therefore cause unforeseen troubles. Due to the general quantum physical limitations to investigate the state of qubits without destroying superpositions, it is generally a challenging task to detect errors. QEC covers the detection of errors without destroying desired superpositions and the error correction either in postprocessing or by correcting the error in a suitable moment of the circuit execution. QEC repetition codes encode a state which should be protected on multiple qubits, and compare the states between those qubits to detect deviations among them, but without directly measuring their state.

To shed more light into the topic of quantum error correction and the underlying processes at work, we propose a specific application-benchmark algorithm based on a novel quantum error correction code developed specifically for this purpose. This code is called the anisotropic repetition code (ARC). It is implemented as part of the Qiskit-QEC framework [12]. The ARC offers the unique benefit that qubits are encoded in different bases and lay next to each other, which is very common among more complex QEC codes. Unlike the repetition codes, these more advanced QEC codes like the surface code have the potential to be used for Fault Tolerant Quantum Computing, but they are very hard to study due to their higher requirements regarding layout and gate error rates. The ARC combines the simplicity and low requirements regime of repetition codes but is susceptible to error and noise which are induced by using different bases at the same time close to each other. This makes it an ideal candidate for benchmarking and

diagnostics towards fault-tolerance. Additionally, useful information can be gathered from the syndrome data, which can be used to derive the properties of the hardware during runtime and to optimize the quantum error correction system itself by tuning the decoding graph. The ARC is not suitable to be used as an all-in-one quantum computing benchmark. It is more of a complement to the whole application benchmark suites like the one from the QED-C, whose authors specifically mentioned quantum error correction-specific benchmarks as a missing piece in their suite [9].

1.2 Definition of the terms Benchmark and Diagnostics in the context of quantum computing

As already mentioned in the introduction to quantum benchmarks in general, there are many benchmarks and benchmarking suites proposed, developed, and available to date. Some of them incorporate many different algorithms, some use proxy algorithms to assess the performance for a certain group of algorithms and other benchmarks use highly randomized circuits, for the sole purpose of benchmarking a device to give an estimate of how any random task would perform.

To disentangle this great diversity of benchmarking tools, protocols, and proposals, Mirco Amico et al. (2023) proposed a definition of a benchmark, which rules it should follow, and what is rather described as a diagnostics method [13]. In a short summary of the definitions, a benchmark informs on the average performances of a device on a randomly chosen task to be comparable over different devices, platforms, and generations, while a diagnostic method gives insights into the performance of a device for similarly structured problems. Furthermore, they propose a universal set of rules, that a benchmark must obey. Those are:

- Random: A test should have a randomized component, like random parameters or circuit inputs, and the output should be an average over these random results
- Well-defined: A benchmark should have a clear role set that defines a protocol or procedure, and leaves no ambiguity, so that it can be closely reproduced
- Holistic: The benchmarking result should be indicative of the performance of a large set of device attributes in as few metrics as possible
- Platform independent: The protocol should be executable on any device connectivity and with their native gates

In their example, the before mentioned application oriented benchmark from the QED-C adheres mostly to these rules. It is a repository of a wide variety of diagnostics methods, which executed all together make up a valid benchmark.

Under this definition, the QEC Benchmark is a diagnostics method being highly indicative of the performance of a large group of quantum error correction codes, and it is developed towards the goal (among others) to be a candidate to be taken up as part of an application benchmark like the one from QED-C.

The exact proposal of the protocol under which it can be considered as a candidate for larger application suites is described in the result Section 3.12.

1.3 Using the ARC for Quantum Error Correction Diagnostics

For the implementation of the quantum error correction diagnostics method, a novel kind of repetition code was developed. It's called the anisotropic repetition code (ARC). The standard repetition code stores the quantum information on multiple other qubits, either in the Z basis for the bit flip code or in the X basis for the phase flip code. In the Z basis a 0 is encoded with $|0\rangle^{\otimes d}$ and a 1 with $|1\rangle^{\otimes d}$. Bit flips can be detected with stabilizer generator-based parity measurements [14, 15]. Parity measurements are performed with the help of ancillary qubits (ancilla or AQ) which are placed between two data qubits (DQ). The ancillas compare the state of their neighboring DQs with controlled not operations (CNOT or CX), which is a 2-qubit gate that changes the state of the ancilla from $|0\rangle$ to $|1\rangle$ (or $|1\rangle$ to $|0\rangle$) if the connected qubit is $|1\rangle$. Performing two subsequent CX with two different data qubits would yield a $|0\rangle$ state on the ancilla. Either both data qubits are $|1\rangle$, then the ancilla flips twice, or both are $|0\rangle$, then it never flips. If only one data qubit is $|1\rangle$, the ancilla flips once and stays in $|1\rangle$, which means, one of the two qubits has an error. When measuring all data qubits with ancillas, it yields precise information, which qubit suffered from an error. 0 and 1 in the phase flip code are encoded with $|+\rangle^{\otimes d}$ or $|-\rangle^{\otimes d}$ and phase flips can be detected. d is the number of data qubits, on which the quantum information is encoded. For the standard repetition code, this is equal to their distance d. For other more advanced QEC codes like the surface code, or color codes, this might not be true anymore. So in the standard repetition code, either a bit flip or a phase flip on a single data qubit can be detected, depending on in which basis the information was encoded. However, bit- and phase flips can not be detected at the same time.

With the ARC, the state is encoded in two different bases, for example, the X and the Z basis (it might as well be X and Y or Z and Y). A logical state $|0\rangle_l$ is encoded with $|0,+,0,+,...\rangle$ and a $|1\rangle_l$ is encoded with $|1,-,1,-,...\rangle$. With this configuration, the anisotropic repetition code would be encoded in the basis ZX. Nevertheless, it is not possible to detect a bit flip and phase flip at the same time on a single data qubit, because a single qubit is still encoded either in the X basis or in the Z basis. Just the neighboring data qubit is in the other basis encoded. The ARC's advantage over the bit flip and the phase flip code lies in the fact, that qubits encoded in different bases lay next to each other, which is also the case for more advanced QEC codes like the surface code or color codes. As with other stabilizer-based QEC codes, the ARC also relies on mid-circuit measurements for syndrome extraction and optionally on resets. The measurement results of the ancilla qubits are called a syndrome and syndrome extraction is the process to measure the ancillas, to deduce the errors in the data qubits. Mid-circuit measurements are measurements that are performed on qubits during the execution of a circuit. Usually, the qubits are just measured once at the end of the circuit. But for

QEC it is crucial to know when an error happened, therefore the ancillas are measured multiple times during a circuit execution. The sequence of one ancilla performing a CNOT operation on two qubits, with the following measurement of the ancilla is called one QEC cycle. The number of QEC cycles for one circuit execution is denoted as T. A reset is a gate to bring a qubit back to the ground state, no matter in which state it was before. Measurements and resets are both very long and error-prone operations compared to single or two-qubit gates. Summarizing this, QEC codes are challenging circuits, mid-circuit measurements are already an advanced feature of a QC system, and it encodes neighboring qubits in different bases like the more advanced QEC codes.

The ARC combines both, the more advanced basis encoding and the simplicity of a standard repetition code, which makes it a good candidate to be used as a diagnostics method with the goal to complete application-suit-based benchmarking tools. Repetition codes are already a very popular code to demonstrate the capabilities of a quantum computer and compare them amongst each other with their performance of executing them over certain distances and numbers of repetitions and their error rates and fidelity [6, 16].

Moreover, since the ARC's purpose is to serve as a diagnostics method, circuits are executed multiple times, with different bases. This makes single qubits susceptible to both, bit flip and phase flip errors. With this method, insightful statistics on the single qubit level can be gained, while the system as a whole executes a quantum error correction code with all the usually necessary gates.

Since measurements are performed on the Z basis, the ARC produces the same output as with the standard repetition codes, therefore standard methods can be applied to analyze the syndromes and recover the original state with a decoder.

As with other stabilizer-based QEC codes, this code can be truncated. This means, with a single circuit execution, data for the largest possible distance is acquired, and in the post-processing, the data qubits can be cut away including the corresponding ancillas to approximate runs with smaller code distances. With datasets for multiple distances d, and multiples numbers of repetitions T the scaling of the information lifetime, and the error rates per cycle can be calculated.

In any case, a benchmark based on quantum error correction codes allows one to assess two regimes. The first one is the macro benchmark, an assessment of the average system as a whole. Usually, the quantum information lifetime or fidelity is measured, so the time or number of rounds that can be reached, where the encoded quantum information is still recoverable. Those are known as standard quantum memory experiments.

The second regime is to measure different error rates on the single qubit level, based on the syndrome measurements. This information is further useful to assess the quality of the qubits and to create enhanced error models for decoding. Further applications of this single qubit statistics can be used to investigate leakage. When a qubit leaves the computations subspace and gets excited beyond the desired energy level of the system this is called leakage. Usually, a quantum computer works with a two-level system and uses the ground state and the first excited state. When the qubit gets excited to the second, third, or any higher excited state, this qubit leaks. This can result in detrimental

effects not only on the leaked qubit, it might also spread and influence the qubits that interact with the leaked qubit. Effectively, it was observed that a leaked ancilla will yield random outcomes on the following mid-circuits measurements. Furthermore, performing a CNOT with a leaked ancilla qubit yields a randomized data qubit state, and a higher-level energy state on data qubits can travel to connected ancillas [17, 18]. Leakage tends to have a big impact and will produce both, time and space-related error clusters. The usual QEC decoders are optimized to handle randomly occurring bit or phase flips, and not correlated error zones.

1.4 Decoders and their optimization with micro benchmarks

An important part of the quantum error correction system is the decoding of the syndromes to detect which class of errors occurred to deduce which data qubit(s) need to be corrected after the readout. A straightforward solution to this problem is to have a lookup table, in which the syndrome measurement patterns are stored, and to which error class it corresponds and how to correct it. With increasing code distances and the number of repetitions, this lookup table would grow too large to be useful. This is why there are other possibilities to decode this information. The general decoding of quantum error correction code syndromes is proven to be NP-hard [19], so this is a topic that deserves serious discussion. The general structure of the decoding process is usually the same. There is an error model, which formulates the possible errors that can occur in a certain quantum error correction circuit. Based on this model, a graph can be constructed, where the nodes stand for a measured ancilla in one QEC cycle and the edges for the relationship between those. Edges are closely related to the data qubit, which are measured with two ancilla qubits. The edges can have weights, which helps to take into account the error rate of the different qubits or gates to help the decoder to better prioritize the underlying errors if multiple possible explanations are found which can explain the syndrome measurements. Those edge weights can be calculated from the before mentioned micro benchmarks.

1.5 QEC diagnostics with syndrome-based analysis methods

In this work, the ARC as QEC code was used to create syndrome data on which diagnostics analyses can be performed. A method is proposed to describe the technical capabilities of a quantum computer system, which is related to the volumetric analysis [4, 20]. These plots are common practice for quantum computing benchmarking.

Therefore, the volumetric-inspired plot gives a quick overview, of which circuits with parameters d and T ran and did or did not pass the success criteria, or where the QC controller hit limitations and the run failed without giving any data to analyze. A method was developed to assess the scaling of the quantum information lifetime ρ based on the code distance d and its intermediate results T_s, which the number of QEC cycles T that are possible to execute and pass the success criteria P_s. Another method to assess the

scaling capability was used, namely to calculate the error rate per cycle ε_l, and its scaling factor Λ.

Further analysis methods were used to put the initial results into context or to perform deeper analyses like to predict the amount of leakage and a more rigorous method to assess this leakage prediction method.

Microbenchmarks were developed, to assess the error rate per qubit, and they were successfully used to optimize the decoding graph.

Different models were tested, which can fit the logical error rate for time series of varying code distances. The initial model just takes stochastic Pauli noise into account, and the fits suboptimally fit the data. Therefore a further model was tested which considers coherent errors in the form of over-rotations. This yields better results in a certain analysis window but fits even worse in other regimes. The optimal model was not found, since leakage seems to be a major driver for logical errors.

To get a handle for leakage, a simple method was developed to detect leaking qubits based on syndrome data. Another method was developed to readout leakage states from the quantum computer and automatically discriminate ground state, first excited states and leakage states from raw readout data of the QC system.

Different parameters were assessed, which can influence the outcome of the diagnostics method. Namely, the difference between using the XZ bases or the XY bases. The influence of resets after the syndrome extraction is investigated. The decoders PyMatching [21] and one described by Bravyi and Haah [22] were compared.

The diagnostics method, along with the other analysis methods were successfully applied to many different IBM quantum computers, with the Falcon processor of 27 qubits (allowing distances up to 11), the Eagle processor with 127 qubits with which distances up to 32 could be analyzed, and even the Osprey processor with 433 qubits was used to execute the to-date biggest distance $d = 148$ repetition code. For these large repetition codes, the choice of the decoder and its optimization level makes a difference of multiple days in decoding time.

Chapter 2

Theory

2.1 Anisotropic Repetition Code

The Anisotropic Repetition Code (ARC) was invented by this projects' supervisor James Wootton and is implemented as part of the open source QEC framework Qiskit-QEC[12]. The ARC is closely related to the standard repetition code. While in the standard repetition code, the data qubits are encoded in either the Z basis or the X basis, the ARC stores the information in both bases. The basis is alternated from one data qubit to the other. Therefore, a logical zero $|0\rangle_l$ and a logical one $|1\rangle_l$ in a distance 5 ($d = 5$) ARC in the ZX are encoded in the following way:

$$|0\rangle_L = |0, +, 0, +, 0\rangle \tag{2.1}$$

$$|1\rangle_L = |1, -, 1-, 1\rangle \tag{2.2}$$

The definition of which qubit is assigned to which basis is denoted as *coloring*. For example, the first qubit at the position 0 in $|0\rangle_L$ is assigned to the coloring group of the Z basis, hence it is a 0, and the second qubit is assigned to be in the X basis group, therefore a 1 and the third again in the Z base and so on.

Figure 2.1 shows the circuit of one round of a distance 5 ARC as denoted in equation 2.1.

To detect errors in data qubits, stabilizer measurements are used. Note, that for stabilizer measurements the bases which are not Z are rotated back to Z to have a common measurement basis.

In this case, the stabilizer generator $M_1 = Z_1Z_2$ can be used to map the states to the $+1$ or -1 eigenspace and therefore act as syndrome measurements between DQ_1 and DQ_2, through AQ_1. With $M = M_0M_1M_2M_3$, the stabilizer operator results in $M|0\rangle_L = |0\rangle_L = +1$ the $+1$ eigenspace and no error was detected. With $MX_1|0\rangle_L = -X_1|0\rangle_L = -1$ the stabilizer maps to the -1 eigenspace, when an error occurred. Note, that when such a circuit is run on a quantum computer with Qiskit, the output of the syndromes is denoted in binary form. So the $+1$ eigenspace is a 0-bit and means no error occurred, and the -1 eigenspace is a 1-bit and flags an error.

Based on the result of the syndrome measurements output, a decoder can then deduce, which qubit suffered from an error.

During a stabilizer measurement the ancilla qubit compares two neighboring DQs with each other, without actually performing a measurement of their states. It is instead done

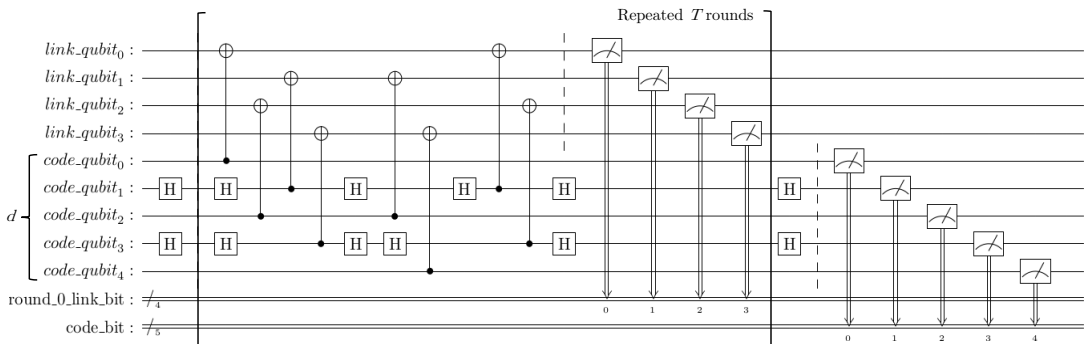


Figure 2.1: Circuit of a distance 5 Anisotropic Repetition Code (ARC). Note, $link\_qubit_n$ are ancilla qubits, and $code\_qubit_n$ are data qubits.

by performing a CNOT gate. The CNOT gate flip the state of the ancilla depending on the state of the control qubit, in this case, the DQ. If a $|0\rangle_l$ is encoded, both DQs should be $|0\rangle$, and therefore the AC never flips. If a $|1\rangle_l$ is encoded, the AQ flips to $|1\rangle$ when performing the CNOT on the first DQ but flips again back to $|0\rangle$ when performing a CNOT with the second DQ. So in both cases, if the DQ is in the same state, the AQ is 0. If one DQ is in $|1\rangle$ the AQ flips once and stays there. The mid-circuit measurement will yield a 1. After the readout of the AQ, there are two options. Option one is that the AQ is reset to the ground state, which is a long and error-prone operation. The second option is to omit the reset gate, and the AQ starts the next QEC round with a $|1\rangle$. If the error persists on the DQ the AQ is flipped back to $|0\rangle$, and in the third round again back to $|1\rangle$. If no resets are used, this needs to be accounted for in the post-processing of the syndrome data, such that as soon as an error was detected, the AQ flip back and forth for the rest of the circuit execution. If the AQ finally stops to flip, the reason might be that the DQ flipped back and therefore resolved the issue, or the other DQ also suffered from an error. What happens then, is described in the section of the Decoding 2.7. Both methods, using resets and not using resets have their advantages and disadvantages. The advantage of resets is, that leakage errors can not spread as fast. The disadvantage is, that it prolongs the circuit execution by a approximately 1/3. This opens the opportunity for the DQs to suffer from more errors or to thermally relax. Furthermore, for the QC controller hardware, a reset operation uses a lot of memory and therefore shortens the amount of possible QEC rounds drastically.

Based on its coloring assignment, the error is identified as a bit flip if the qubit was on Z basis or a phase flip if it was in the X basis. If the qubit was in the Y basis, it could have been a bit flip or a phase flip.

There are multiple possibilities to map an ARC onto a given Qubit connectivity layout. The IBM quantum computers used in this work all use the heavy-hex layout. In this project, the ARC was always used in a linear chain of alternating DQs and AQs as depicted in Figure 2.2 a). This linear chain has the possibility to close the loop in some cases, by adding an additional AQ which connects the last and the first DQ, as depicted in dashed lines. In the 2D layout, additional AQs can be introduced such that some DQs

are compared against three other DQs as shown in Figure 2.2 b). On the other hand the 2D ARC layout poses some more restrictions towards the connectivity of a quantum computer. Therefore, the linear chain layout is more suitable to be used in benchmarks and diagnostics because it is simpler to be created automatically and randomly, and it makes it more comparable throughout different QC connectives.

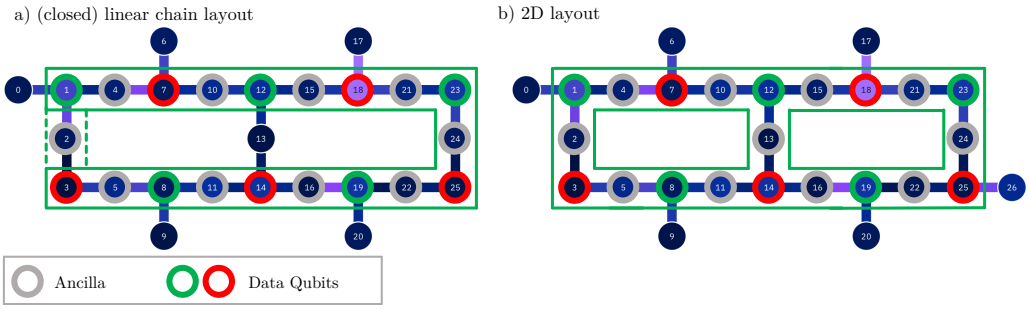


Figure 2.2: Different layouts of the ARC, based on the IBM Falcon processor connectivity. a) the linear chain, optionally with a closed loop (dashed lines) b) the 2D layout in which the data qubits are measured 3 times with parity measurements (with some exceptions)

Usually, 4 differently parameterized circuits are executed during one run of the ARC on a quantum computer. The first two encode a $|0\rangle_l$ whereas the first circuit uses one bases combination, like XZ and the second circuit reverses the order of the bases to ZX. The third and the fourth circuit encode $|1\rangle_l$ again with changing bases. Technically, all four circuits are executed directly after each other once, separated by a short time period called the 'rep_delay'. The rep_delay is used as a hard reset, to allow every qubit to relax into the ground state. This procedure is then repeated many times over in so-called shots. Throughout this work, usually, $s = 4000$ shots were used.

The main experiment executed with the ARC is a QEC memory experiment. The goal is to encode a state with the data qubits and measure how long this state persists or can be recovered while executing the QEC protocol. For ARCs executed with identical parameters (d, T, resets, bases, quantum computing backend, layout), the execution time stays the same and can be calculated, by calculating the time needed for one QEC cycle, times the number of repetitions. Varying numbers of d, do not have an influence, because the gates and readouts of different data qubits and ancillas are done in parallel. The choice of the bases has a very limited influence on the execution time because only the number of single-qubit gates are different, and the single qubit gates are very fast, compared to readout and reset gates. Therefore running a repetition code for multiple different repetitions T, for which the time in seconds can be calculated, the lifetime of the encoded quantum state can be determined.

Each run needs to pass the success criteria, that the encoded initial state can be recovered. This is calculated by the logical error rate (or fidelity) which has to be below

a certain value, P_s.

2.2 Calculating the physical error rates

The physical error rate is the rate of errors on the data qubits. The state of the final readout of the data qubits of one ARC circuit execution is compared to the initially encoded state. Therefore, all the syndrome bits of the ancilla qubits are ignored and only the readout bits are taken into consideration. Afterwards the ratio of wrong bits vs all bits is taken to deduce the error probability.

This must not be confused with the logical error rate. The logical error rate is the error rate of the logical qubit, after the QEC correction procedure was executed and the corrections were applied.

This ratio of the measured uncorrected states of the data qubits versus the initially encoded states is taken for varying numbers of QEC cycles T of the ARC circuit to extract different error probabilities for varying QEC circuit execution durations. The goal is to obtain the error rate per qubit over time, to get a sense of how much information is lost over time.

When $e(i, T)$ is the number of wrong readout bits for one qubit i for one circuit execution and s is the number of total bits (which corresponds to the number of shots for one circuit run), then the physical error probability per qubit is calculated as follows:

$$p_e(i, T) = \frac{e(i, T)}{s} \tag{2.3}$$

s is fixed through all experiments to $s = 4000$, as long as no post-selection was performed.

2.3 Determine quantum information lifetime on physical qubits during QEC operations

Knowing the quantum information lifetime of physical qubits during QEC correction operations is beneficial because it provides insightful information about the data qubits during quantum error correction. This information is used to debug the QEC procedure. For example, assessing whether high logical error rates (or weird patterns in the logical error rate) stem from errors in the physical qubits, or from the decoding procedure. The calculation of the physical qubit error rate was already explained in Section 2.2.

To get from the physical error rate to the quantum information lifetime of the data qubit, one needs to consider the duration of the QEC cycle. IBM quantum computers have an internal clock system and gate execution is controlled with the controller systems' internal clock cycles. The number of clock cycles (cc) needed to perform one round of the ARC is dependent on quantum computer hardware, the supported native gates, and therefore the circuit compilation. For example for ibmq_kolkata, it takes around $cc \approx 42000$ clock cycles for one QEC cycle T, and each clock cycle takes $dt = 2.222 \times$

10^{-10} seconds. With this information available, the duration of each circuit execution is calculated, and combined with the physical qubit error rate the information lifetime is retrieved. The total duration of one QEC cycle is denoted τ_c and is calculated by multiplying the number of clock cycles with the time of one clock cycle in nanoseconds: $\tau_c = cc \times dt$. The physical qubit lifetime is determined by fitting the physical error rate for each run of varying QEC cycles T with a formula which is known for calculating $T1$ relaxation time of qubits [23].

$$p_e(i, T) = Ae^{\frac{-T\tau_c}{\tau_p}} + B \tag{2.4}$$

τ_p is the quantum information lifetime based on the physical error rate. τ_p is expected to be shorter than the official T_1 times provided by the IBM Quantum backend. $T1$ thermal relaxation numbers are created with regular runs of calibration and characterization protocols. The difference to τ_p is that the data qubit experienced rotation- and CNOT gates, and idled for some time while the ancillas are read out. On the other side, during the ancilla readout, they undergo dynamical decoupling, which is a strategy to mitigate phase shifts. The qubit is rotated to the X-plane, and accumulates shift, in the middle of the sequence the qubit flips, such that the accumulated shift is reversed and then it is rotated back to Z. It has been shown that dynamical decoupling is beneficial for idling qubits, whose information should be kept [24]. Note, dynamical decoupling is an optimizaton technique, which needs to be reported when used in benchmarking and diagnostics [13].

2.4 Break-even point analysis

The break-even point is the point where a specific quantum error correction system does more good than harm. When developing QEC systems, it is important to study their scaling properties. It is also important to evaluate if the QEC system on a specific quantum computing system can beat the quantum information lifetime of idling physical qubits [16, 25, 26].

QEC exists to protect quantum information over time usually by distributing the information to many more qubits, and to detect errors and correct them with gates that are themselves error-prone. Therefore, the question arises if QEC can also account for the newly induced errors and still outperforms a single qubits performance in keeping the quantum information.

To analyze the break-even point to deduce if QEC is overall beneficial for a specific QEC code on a certain system, the average lifetime of single qubits is compared to the lifetime of a logical qubit.

For the analysis, the thermal relaxation times of single qubits are needed, and the logical lifetime of a state protected by QEC correction. T1 and T2 are given by the IBM Quantum backend and were acquired with experiments as described here [23, 27, 28].

The logical lifetime is calculated by using the logical error rate P_e (see Section 2.9) and the duration of the QEC cycles for different τ_c. These points are fitted to deduce τ_l,

the logical lifetime in seconds.

$$P_e = Ae^{\frac{\tau_c T}{\tau_l}} + B \tag{2.5}$$

To analyze where the ARC stands on the current hardware regarding the break-even point, the average of T1 and T2 are plotted against time, together with the logical lifetime. The results can be found in Section 3.1.

2.5 Leakage analysis

Leakage states are states that are outside of the usual computational subspace. Usually, quantum computers work with a 2-level system, using the ground state and the first excited state. But quantum systems are not binary, therefore it is possible that states are excited beyond the first excited state and end up in the second excited state. Since the controller hardware and the algorithms do not expect qubits to be in the second excited state, they can not handle this and it leads to errors.

2.5.1 Estimate the leakage based on syndrome data

To estimate leakage from the syndrome data, the number of detection events within the syndrome strings is counted. In the case of running the ARC with resets enabled, a 1 in the syndrome string stands for a detected event, an error. A 0 denotes no error. In the case of no resets, the first detected error event is denoted as 1. When this error persists, in the next QEC round the AQ is flipped back to 0. Due to the CNOT, the error causes the AQ to flip between 0 and 1 in every QEC cycle. In any case, with and without resets, the counted number of 1's in the syndromes should not exceed the number of 0's, in the limit big T, under the assumption, that the logical error probability converges to $P_s = 0.5$.

The leakage analysis is taking these effects into account and analyses the statistics. The number of 1's in all syndromes is counted and divided by the number of shots. If the rate of 1 is bigger than 0.5, this is an indicator of leakage. A leaked state readout is usually interpreted as 1 by the quantum control hardware, and leakage does not dissolve so fast. Therefore, it is expected to see an accumulation of 1's on the level of a single ancilla qubit. This analysis is executed for every circuit run and plotted against T. At this moment, this analysis was only used, to visually inspect qubits and find explanations for unexplainable behavior during a QEC run. A method to quantify the leakiness of a qubit might be, to calculate the area under the curve, above 0.5 for every qubit. Together with the level 1 readout as described in section 2.5.2, this method of quantifying leakage with just the QEC syndrome bits could be verified within this thesis.

2.5.2 Detect leakage through optimized readout processing

The current control hardware of the IBM quantum computers does not support the direct readout of leakage states, which are second or higher excited states. Nevertheless, it can

be extracted manually, which is exactly what was done in this thesis, to see if leakage is the source of some unexplainable data. One way to come by with leakage is to modify the process of the readout of the measurement results of the IBM quantum computers, to support the detection of $|2\rangle$ states. This is done by starting the job in Qiskit with different parameters, to get level 1 results. With this modification, the system returns one point for each shot in the imaginary plane. Depending on the position of this point, it is most likely a $|0\rangle$, or a $|1\rangle$. Usually, the IBM quantum systems make this discrimination for the user with optimized parameters to give highly accurate results, but it does not support $|2\rangle$. Therefore, a modified discriminator was developed. A Gaussian Mixture Model (GMM) was used, which has the ability to learn a discriminator unsupervised.

Figure 2.3 shows the 2D plane, with one dot per shot, for the readout of one qubit. Two levels of discrimination were executed. The first distinguishes the two big blobs of points, whereas the $|0\rangle$ states are usually on the left side (blue), the $|1\rangle$ states are to the right. The second stage of discrimination is run inside the space identified for the $|1\rangle$ and tries to find again two kinds of separated point clouds. Therefore, the upper / upper-left part should contain $|2\rangle$ states (green) and the lower part contain the real $|1\rangle$ states in red. To evaluate if the second discrimination run needs to be executed, the spread of the points in the y-direction is calculated. If the spread of the $|1\rangle$ states is bigger than the one of the $|0\rangle$ states, it is likely that there is leakage, and the second discrimination is executed to find these states.

With this discrimination and detection of leakage states, the usual bit-strings can be reconstructed, but this time, including 2 for readouts that are likely leakage. With this first step established, there are multiple ways how to use the information about leakage states. The first and most simple one is to post-select only shots that do not contain any leakage states, and evaluate how much the logical error probability drops. The rationale behind this is, that only shots are taken into account which are certainly 0 or 1. A 2 indicates leakage, and the bare existence of leakage has many implications. The Google Quantum AI QEC team (2021) observed, that leakage states can influence other nearby qubits, depending on the state of these qubits. If qubits are in 0, leakage has likely no effect, but if it is 1, this can change the state of these qubits randomly [17, 18]. Therefore, just the detection of one leakage state in one shot indicates that many qubits suffered randomization. Instead of trying to correct them, the whole shot is ignored. Another approach makes use of the knowledge of leakeded qubits to modify the decoding graph accordingly, and sets the edge weight of the nearby qubits to 0.5 to indicate, that those qubits are likely randomized. This technique is more challenging to implement, but it was successfully used in other projects [18, 29].

2.6 Calculating the edge weights for the decoding graph

A decoding graph can have uniform edge weights, which assumes that every possible error which can be expressed by the decoding graph happens with equal probability. This is a valid solution and serves as a basis for decoding if no more information about the error rates are available. If more information about the possible errors is available,

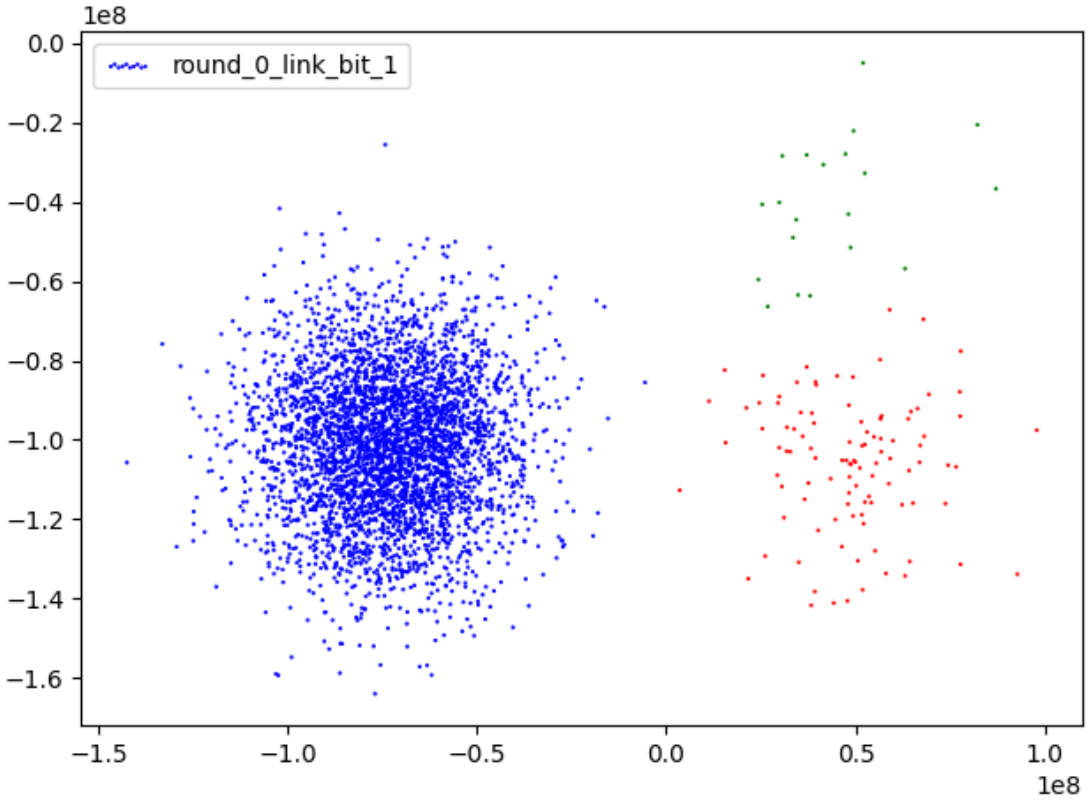


Figure 2.3: Example discriminatation of the readout shots, depicted in the complex plane. The blue dots are discriminated as $|0\rangle$, the reds as $|1\rangle$ and the green ones are likely to be leakage states.

the decoding graph can be tuned with different edge weights which better map the error probabilities of the actual quantum computer and its qubits.

Those edge weights are calculated using the syndrome information of a repetition code. Every 1 in the bit string of the syndrome stands for an error (assuming no resets are used). There are multiple possibilities to calculate those edge weights, and throughout this work, two different methods were used. The first method is a simple approximation. The syndromes are analyzed for errors and assigned to the nodes in the decoding graph, whereas every edge is then analyzed if their nodes contain errors. A ratio between having errors on both nodes and having no errors at all in the nodes is then determined and assigned to the edge. The second more sophisticated method was developed by Spitz et al. and is described here [30].

2.7 Decoders

The output of running a QEC circuit consists of final readout bits from the data qubits and syndrome bits from the readout of the ancilla qubits. The syndrome bits flag up a 1 when the two measured data qubits are not the same. If there are all zeros, no error was detected. In the readout of the code qubit part, everything is fine, if there are all zeros or all ones, this depends on in which state the data qubits were initially encoded.

The following few steps are predominantly the same for the decoding algorithms. It consists mainly of creating another syndrome round out of the final readout bits and then generating a decoding graph, in which the nodes represent the syndromes. The decoding algorithms work on this graph to find the sources of errors based on different strategies.

The output coming from the IBM Quantum backend looks in the following way for $T = 1$ and $d = 5$:

$$\text{output} = \text{code bits} + T \times (\text{ancilla bits}) = 0_4 0_3 0_2 0_1 0_0 \; 0_3 0_2 0_1 0_0 \tag{2.6}$$

The first processing step is to compare the two neighboring code bits with each other, and if they are equal, flag it with zero, and if they are not equal flag it with one.

When this parity of the final readout of the data qubits is established, it is not necessary to keep all the final readout bits anymore, just one of them is enough. The parity information tells how trustworthy the readout is and the decoder works out if the readout bit needs to be flipped.

This changes the output of the QEC circuit from step 2.6, adds another round of syndromes, and reduces the final output to one bit, denoted with subscript r. This is shown in step 1 in (2.7):

$$\text{step 1} = \text{readout bit} + \textit{parity}(\text{code bits}) + T \times (\text{ancilla bits}) = 0_r \; 0_1 0_2 0_3 0_4 \; 0_1 0_2 0_3 0_4 \tag{2.7}$$

Based on the parity and the syndrome bits, the next step is to create a decoding graph with dimensions $d \times (T + 1)$. The different syndrome measurement bits are placed on the x-axis, and the number of repetitions T is on the y-axis of the decoding graph, as depicted in Figure 2.4 for $d = 5$ and $T = 1$. To account for the data qubits on the boundary of the chain a virtual node called 'b' is added to the decoding graph. The decoding graph for this example of $d = 5$ and $T = 1$ is depicted in Figure 2.4.

The edges in the graph represent the data qubits. When two neighboring nodes flag an error, then usually the data qubit of the connecting edge suffered from an error. If the connecting edge is a horizontal line it means that the error was introduced before the whole syndrome measurement operation was executed. If the edge is a diagonal line, it indicates that an error happened in between a syndrome measurement operation, namely after the first CNOT to one data qubit was executed, and before the second CNOT to the other data qubit was executed, such that the error is only detected when the first qubit in the next round $T + 1$ is checked with a CNOT as depicted in Figure 2.5 d). A vertical line connecting two nodes indicates a readout error of the ancilla.

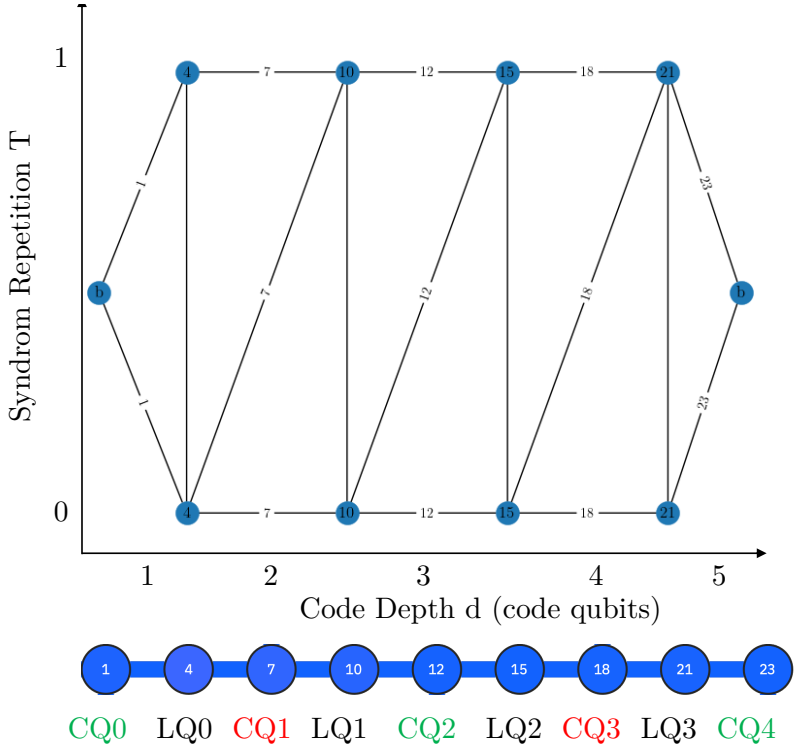


Figure 2.4: Decoding graph of a $d = 5$ ARC over $T = 1$ rounds. Shown below is a layout of how this ARC code is mapped to an ibmq_kolkata Quantum computer from IBM Quantum, to show the relation to the decoding graph. CQn depicts the nth code qubit, the coloring green means the Z basis and red means the X base. Ancilla qubits (AQ) are depicted in black.

The second step is to deduce the nodes in the graph that represent changes in the syndromes. They can be seen as the boundary of groups of zeros and ones. The rationale behind this is that errors on two neighboring DQ are not detected as errors by the AQ comparing them. The errors are detected by the AQ on the other side, when compared to a third DQ, that has supposedly no error. Therefore, just the boundaries of areas with and without errors are marked by the AQ, and it is a matter of the decoder to figure out, which area is the one containing the errors, and which one contains the nodes with no errors. Considering Figure 2.5, it might be difficult to imagine how such areas look like, but usual decoding graphs are bigger. The biggest repetition code executed in this work is $d = 148$ and $T = 60$, therefore this graph would not fit on this page. Estimates for the distance for QEC codes that provide real fault tolerance for a near-unlimited number of repetitions are between 1000 and 10'000 data qubits.

In the following Figure 2.5 different errors were introduced to the ARC circuit together with the decoding graphs which show, how these errors manifest in them. Note, that the circuit contains all three errors, but the circuit was executed for each error individually, to create the decoding graphs in Figure 2.5 b) - d).

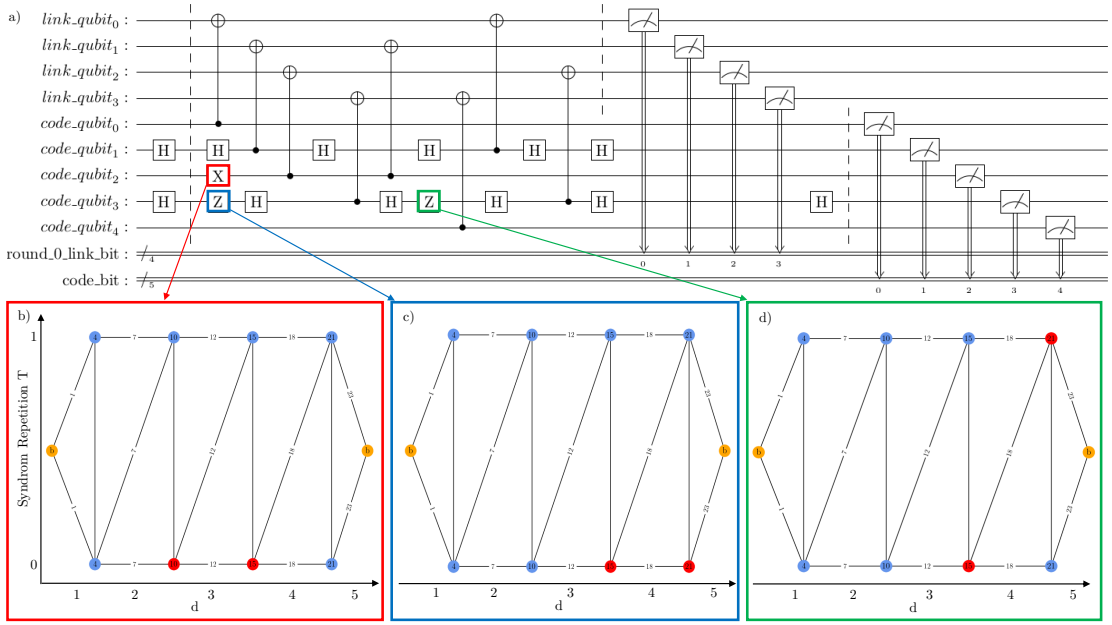


Figure 2.5: Three different kinds of errors are shown and how they flag up in the decoding graph. Note, even though all three errors are drawn in one circuit, the circuit and the analysis was run for each error individually. a) Induced bit flip on $code\_qubit_2$ (real qubit id 12). b) Induced phase flip on $code\_qubit_3$) which can be detected, because this code qubit is encoded in the X basis. c) Induced phase flip during mid-circuit measurements. This happens when an error occurs between two CNOTs. Therefore, on one ancilla, it is detected in this round, and on the other ancilla it is detected in the next round.

In the following part, the intermediate steps are shown of how such an error ends up as a node in the decoding graph as pseudo-mathematical operations. It begins with the error in 2.5 a). Note, that this example assumes resets after the AQ readout. The error is a bit flip happening on $code\_qubit_2$, which is encoded in the Z basis, and therefore it can be detected. Step (2.8) gives the full output of the syndrome measurement from the quantum computer without any operations done yet. The subscripts after the bits denote the qubit index as shown in the Figure 2.5. It is important to point out, that the output comes in reverse order from the IBM Quantum backend. The first five bits are the measurements of the data qubits denoted as $code\_qubits$ and the second part of 4 bits are the output of one round of parity measurements from the ancillas.

$$\text{output} = 0_4 0_3 1_2 0_1 0_0 \; 0_3 1_2 1_1 0_0 \tag{2.8}$$

In the next step, shown in step (2.9) the first separation of the $code\_qubit$ readout happens. In the second row (2.10), the parity between the code qubits is calculated. The last row (2.11) is the normal syndrome measurements without any changes. Except that

the subscripts are omitted.

$$\text{step1} = 0 \tag{2.9}$$
$$0110 \tag{2.10}$$
$$0110 \tag{2.11}$$

In the step in (2.12) the separated physical bit is omitted, and only the changes between the different measurement rounds T are shown. Note that the 1 marks a change from 0 to 1, or 1 to 0, going from the bottom to top of the block, since the syndrome measurement rounds come in this order.

$$\text{step2} = 0000 \tag{2.12}$$
$$0110 \tag{2.13}$$

In the next steps, those two 1 in the bottom line are converted into nodes. Since the bottom line represents the first syndrome measurement (T = 0) they get assigned $time = 0$, and the node itself belongs to an ancilla qubit, those are the ids 10 and 15 which are real qubit id's from the IBM Quantum Falcon 27 qubit layout (for example in Figure 2.2), and each node stores also the two connecting data qubits, on which the ancilla performed the parity measurement.

The second example in Figure 2.5 c) shows a phase flip on $code\_qubit_3$. Analog to the calculations conducted for the first example, it correctly flags the nodes of $link\_qubit_3$ and 2, which together enclose $code\_qubit_3$.

The last example in 2.5 d) introduces an error between two CNOT gates. The error happens after the CNOT gate was executed from $code\_qubit_3$ to $link\_qubit_3$, but before the CNOT to $link\_qubit_2$ could be executed. Therefore, $link\_qubit_3$ does not flag an error in the first round, but $link\_qubit_2$ does flag it. In the second round, it would be caught by $link\_qubit_3$. Since only one QEC cycle was executed, it is instead directly visible in the final readout of $code\_qubit_3$, and is therefore correctly identified.

After introducing the pre-processing of the decoding, the next step is to feed the decoding graph to one of the many quantum error correction decoders. Two decoders were used throughout this work. The first one is the Bravyi-Haah decoder [22]. Another decoder used is the Minimum Weight Perfect Matching decoder, as it is implemented in the PyMatching python library [21].

Based on the decoding graph and the syndrome bit string, the decoder predicts if the final readout bit requires to be flipped. This final readout bit was separated out in step (2.7).

2.8 Quantum Volumetric plots

Quantum Volumetric plots are used in the quantum computing application benchmark suit by QED-C [9]. To fit the ARC-based benchmarking protocol into this quantum application framework, Quantum Volumetric plots are required. The difference between

the official volumetric plots and this work is, that with a bigger QEC code distance the fidelity is improved, while on non-quantum error correction algorithms, a bigger width of the code reduces the fidelity of the result. They have in common, that more repetitions of the code, or the depth of the algorithm, reduces the fidelity of the result. Standard volumetric analysis gives a square bounded by the data point with the biggest-still possible width and depth and everything below that, or within this square, yields certainly a positive outcome. For quantum error correction codes, an increased width of the code improves the fidelity such that a positive outcome mcan be achived for a larger depth (T). This yields to a triangle of success. On the other hand, distances are limited by the number of qubits available on the chip and the connectivity. The number of successful repetitions, is either bound by too low fidelity or the QC controller system yields an error due to controller hardware limitations.

2.9 Logical error probability

The logical error probability $P_e(T, d)$ is calculated by taking the fraction of correctly decoded shots s_c and the total number of shots s, for each executed circuit, as shown in Equation 2.14.

$$P_e(T, d) = \frac{s_c(T, d)}{s)} \tag{2.14}$$

Each circuit is parameterized by the distance d and the number of QEC cycles T.

2.9.1 Stochastic pauli noise

These data points $P_e(T, d)$ are plotted on a graph with P_e on the y-axis over T in the x-axis, with different traces for each distance d.

A model to fit those points, is often used in this context [22, 31, 32]:

$$P_e = \frac{1}{2}(1 - (1 - 2\varepsilon_l)^T) \tag{2.15}$$

whereas ε_l is the logical error probability per QEC cycle.

Note, this model only accounts for stochastic Pauli noise. When other error sources are dominant like leakage or coherent errors, this model is not accurate anymore. A variant was developed which accounts for coherent errors in the form of systematic over-rotation. This was also investigated and has proven beneficial in distinct use cases, as described in Section 2.9.2.

2.9.2 Stochastic noise and coherent errors

The formula 2.15 only accounts for stochastic Pauli noise and therefore needs to be modified with a term that accounts for coherent noise, according to [31]. The model is

given in equation 2.16.

$$P(T) = \frac{1}{2}\left(1 - (1 - 2\varepsilon_l)^T\right) + \frac{1}{2}\left(\frac{\bar{\varepsilon}T}{2}\right)^2 \tag{2.16}$$

The second term stems from a first-order approximation for small T of the coherent noise process $\left(sin^2(\bar{\varepsilon}T/2)\right)$. $\bar{\varepsilon}$ is another fit parameter that stands for the rotation angle.

2.10 Scaling factor of the lifetime

The model to fit the logical error rate is applied to calculate the maximum number of QEC cycles, that fulfill the success criteria of staying below the logical error rate of $P_s = 0.3$ for each code distance. In certain cases of quantum computers with low error rates, the maximum possible number of QEC cycles T on a quantum computer did not reach P_s, therefore an extrapolation is required.

To calculate the number T, that fulfills the success criteria, denoted as T_s is displayed in the following formula 2.17. It is derived by rearranging equation 2.15 and setting P_e to a constant $P_s = 0.3$.

$$T_s = \frac{\log\left(1 - 2P_s\right) + 2\pi i}{\log\left(1 - 2\varepsilon_l\right)} \tag{2.17}$$

To derive the scaling of the lifetime ρ, and the base lifetime T_0, T_s is determined for every distance d and plotted. Then equation 2.18 is fitted over those points.

$$T_s = a \times \left[\left(\frac{1}{\rho}\right)^{\frac{d+1}{2}}\right] + T_0 \tag{2.18}$$

The formula is inspired by finding an exponential coefficient, to describe how well the lifetime scales in terms of QEC cycles with the distance.

The underlying assumption is, that the lifetime scales exponentially like $T \approx (\frac{1}{\rho})^{\frac{d}{2}}$, but applied to the found data, it required some adjustments regarding a constant offset T_0 and a linear factor a. In the end, the linear factor a was bound to values between 0.95 and 1.05, because it prevented data comparison. a is kept between these parameters, because the value indicates if the scaling is truly exponential for $a < 1$ or if it tends to have a more linear character with $a > 1$.

2.11 Scaling factor of error rate per cycle

The scaling factor of the error rate per cycle Λ is derived by plotting the logical error rate per cycle ε_l over different distances and fit the values with with equation 2.19.

$$\varepsilon_l = \frac{C}{\Lambda^{\frac{d+1}{2}}} \tag{2.19}$$

where as C is a constant and Λ is the scaling factor. This formula is commonly used to calculate the scaling factor of the logical error rate per cycle [17, 18, 32].

The Lambda factor is specially mentioned as an important metric in the studies of the Honeycomb code, which is one of the more advanced QEC codes for which the ARC is expected to be an adequate predictor [33].

2.12 Truncation of the ARC Code

To calculate the scaling factors ρ and Λ in Equations 2.18 and 2.19 the ARC needs to run for multiple distances d and QEC rounds T. To make this process more efficient, it is not strictly necessary to run circuits for different d. The circuit runs with smaller d can be approximated in a post-processing procedure by cutting away syndrome bits from the data acquired with a bigger distance. After this step, the decoding algorithm is run again to try to recover the encoded state with less syndrome data.

Under the assumption that the noise-induced errors occur independently, this truncation operation is executed and the resulting data should be of similar quality to the data set that would have been acquired by running the circuit on a quantum computer reduced by one qubit and ancilla.

The truncation by one qubit can only occur at the beginning or the end of the syndrome bit-string, either from the left side or the right side. Cutting away a qubit in the middle would separate the code into two separate decoding problems with less than half the code distance for each side, which is not desirable.

As an illustration, the ARC circuit mentioned before with a distance of 5 and one measurement round ($T = 1$) yields an output as in step 1 (2.20). The code bits are a classical register in which the readout result of the Z basis measurement of the *code_ qubits* is mapped to, likewise the *ancilla bits* are mapped to the code register denoted as *round_*$(T-1)$*_ link_ bit*. The bits have their position marked in subscript, to point out that their position in the register is relevant and comes in the reverse order compared to the order in the circuit.

$$output = code\ bits + T \times ancilla\ bits = 0_4 0_3 0_2 0_1 0_0\ 0_3 0_2 0_1 0_0 \tag{2.20}$$

The first group of five bits represents the final measurement of the $d = 5$ qubits. The second group of four bits represents the parity measurements of the $d - 1 = 4$ ancilla qubits.

Figure 2.6 shows the same distance 5 ARC circuit as before, but the truncated readout bits from the qubit and ancilla are shaded in gray. This gray area is cut away with the truncation operation. The truncation operation is abbreviated with $trk()$, which means, that one bit is cut off from the right as denoted in 2.21.

$$trk(output) = trk(code\ bits) + T \times trk(ancilla\ bits) = 0_3 0_2 0_1 0_0\ 0_2 0_1 0_0 \tag{2.21}$$

This truncation is repeated multiple times, if required.

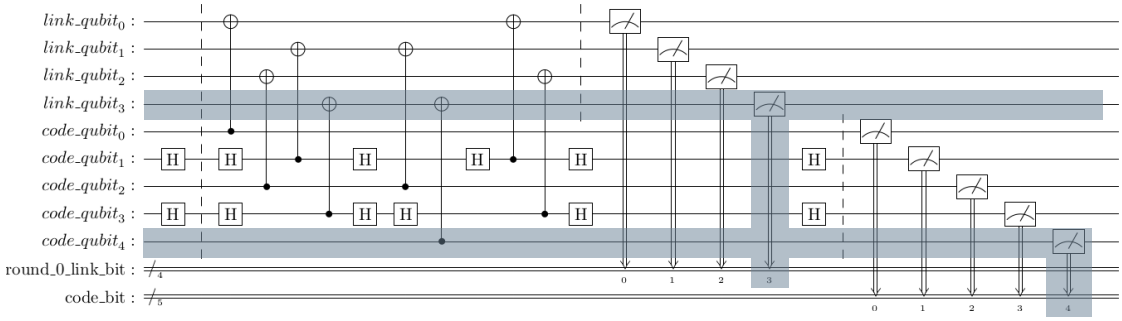


Figure 2.6: Distance 5 Anisotropic Repetition Code with a truncated last qubit and last ancilla (shaded in gray)

There are two strategies to perform the truncation. One strategy is to try to reproduce the quantum computer run as accurately as possible. This is truncating one-sided. For ARC runs in this project of different distances, the qubits are successively added to the end of the chain.

The second strategy is to average a subset of qubits over all available qubits of the biggest distance. This creates more uniform data but does not accurately resemble how the circuits of different distances were executed on the quantum computer. This averaging truncation, also called subsampling, is used by Google Quantum AI to analyze their repetition codes [32]. In this thesis, only the one-sided truncation is examined.

2.13 Noise models for simulations

For the simulations on the Qiskit Aer Simulator within Qiskit Framework [34], noise models are used which derive their error rates from the actual or past QC device properties, on the single qubit, and gate level. To derive the error rate for single and two-qubit gates, the gate times, as well as measured error rates, are taken into account. Additionally, for each single qubit thermal relaxation is added based on the measured T1 and T2 times. Those are built-in functions of Qiskit and allow the simulation of quantum circuits for specific quantum computers.

Chapter 3

Results & Discussion

First and foremost, the recently developed Anisotropic Repetition Code (ARC) is established by discussing the Quantum Error Correction (QEC) break-even point on the current hardware to see if the ARC can outperform single qubit lifetimes, followed by an example of the standard analysis of the lifetime scaling factor ρ and logical error rate scaling factor Λ. The scaling behavior is further compared over a number of current IBM quantum computers. The used Falcon 27 qubit machines are ibmq_montreal, ibmq_mumbai, ibmq_kolkata, ibm_hanoi, ibm_auckland, and ibm_geneva. On bigger quantum computers like the 127 qubit Eagle processor ibm_sherbrooke and the 433 qubits Osprey processor ibm_seattle ARCs were executed with distances up to 148 [35].

Furthermore, the variability of quantum computers over time based on the scaling analysis is performed. The following subchapters discuss the influence of certain parameters of the ARC, namely using different decoders, using different basis paris like XY or XZ and the usage of resets after the syndrome measurement. Having established the influence of the different parameters, a comparison study on many IBM quantum computers is conducted, and put into context to other studies.

All those measurements were carried out by executing the ARC on quantum computers while mapping the full parameter spectrum of d, T, the order of the basis, and the logical states $|1\rangle$ and $|0\rangle$. There are further methods of collecting similar data. The first is to derive the dataset by running a biggest possible distance ARC on the quantum computer and obtain smaller distances with a post-processing procedure called truncation. The second method is a simulation and the results are compared to the data derived from quantum computers.

During this work, other kinds of analyses were conducted. One is to optimize the decoder by using the syndromes data derived error rates per qubit. This analysis is also interesting per se, because it compares single qubit error rates during QEC execution against the data acquired from the official QC characterization and calibration procedures. To examine leakage, a simple syndrome data-based method was developed which is verified by actually measuring leakage states.

The concluding topic is a proposal of a protocol, with which the ARC might be used as a diagnostics method within larger quantum computing application frameworks, like the QEC-C.

3.1 Introduction to the QEC Benchmark with the ARC

First, to introduce the general QEC capabilities of the ARC on current quantum computers the break-even point is analyzed. Figure 3.1 compares the average information lifetime in QEC cycles T and in microseconds (μm) for three different situations. The first situation (green) is the average of the thermal relaxation of the single qubits, without any further gates applied. This is the clean theoretical lifetime of a qubit as presented on the IBM Quantum webpage and the backend [35]. The second curve (blue) is the lifetime obtained from an ARC circuit execution with a distance of $d = 10$ and applied quantum error correction. As a comparison, how much noise is induced by executing the ARC, the lifetime of the physical data qubits during an ARC run is shown in red, but they are not error corrected.

This graph shows that an ARC with a distance of $d = 10$ operates past the break-even point. QEC yields a positive outcome and prolongs the lifetime. Therefore, QEC is beneficial. Executing the ARC induces noise and errors, which leads to a shorter lifetime of quantum information on uncorrected data qubits.

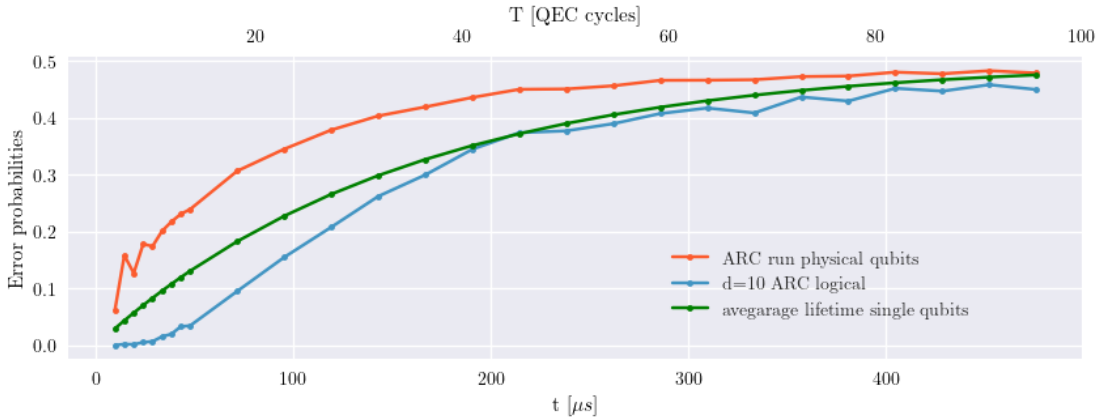
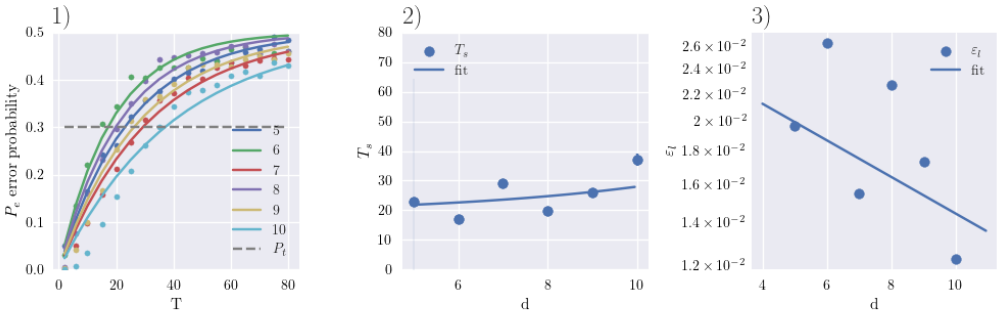


Figure 3.1: Comparison of lifetime decay of quantum information. Shown in green is the error probability of the average of the T1 and T2 times of single qubits. The lifetime of using the ARC including error correction with a distance of $d = 10$ is plotted in blue. The third curve in red shows the lifetime of the data qubits of the ARC with omitted error correction. The ARC with corrected errors is able to extend the logical lifetime over the average physical qubit, but leaving the data qubits of the ARC uncorrected has a detrimental effect on the lifetime.

Similar results were achieved, by executing a standard repetition code on IBM quantum machines already 2018 by Wootton and Loss [36] in the comparison of the repetition code against the single qubit memory.

After establishing how the ARC's performs against single physical qubits, the analysis of the lifetime scaling factor ρ and logical error rate per QEC cycle scaling factor Λ is presented in Figure 3.2 a. This Figure is used as a standard throughout this thesis to analyse the scaling. Therefore, it is here discussed in detail once.

The code distances range from $d = 5$ to $d = 10$, with QEC cycles ranging from $T = 2$ up to $T = 80$, in steps of 5.



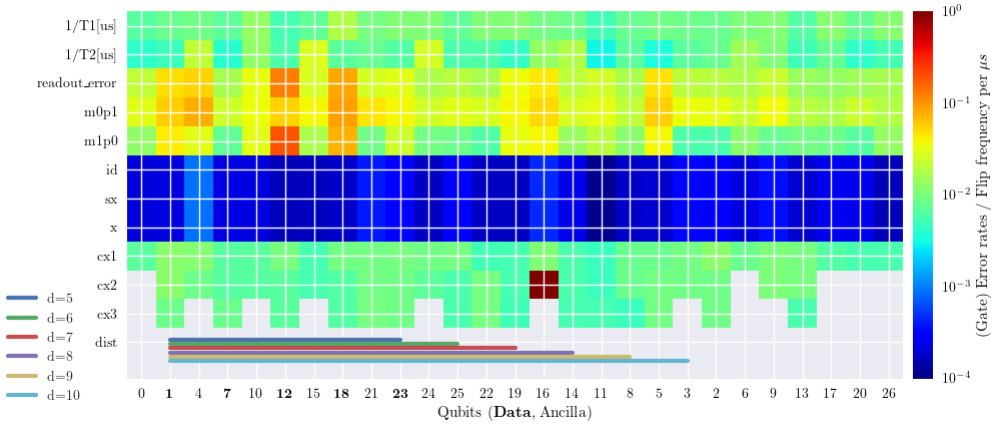
d	T_s	ε_l
5	22.91 ± 0.4	0.02 ± 0.0
6	17.02 ± 0.83	0.026 ± 0.001
7	29.27 ± 0.99	0.015 ± 0.001
8	19.8 ± 0.87	0.023 ± 0.001
9	26.14 ± 1.16	0.017 ± 0.001
10	36.99 ± 1.99	0.012 ± 0.001

ρ	0.659
T_o	18.55
A	0.95

Λ	1.138
C	0.029

ibmq_mumbai - mar 9 bh

a Visualization of one ARC run on ibmq_mumbai on march 9th 2023, of code distances ranging from $d = 5$ to $d = 10$ and syndrome rounds from $T = 2$ to $T = 80$ roughly in steps $T = 5$, decoded with the Bravyi-Haah decoder (bh). 1) shows a plot of the logical error probability P_e for each point d and T, fitted by equation 2.14, and the maximum lifetime T_s that fulfills the success criteria of $P_s = 0.3$ (gray dotted line). T_s and ε_l are extracted and tabulated below the graph. 2) shows a graph of T_s for each distance fitted by 2.18 to the deduce the lifetime scaling factor ρ. 3) plots the logical error rates per cycle ε_l, to derive the Lambda factor Λ, which represents the scaling factor of the logical errors per cycle.



b Qualitative representation of the hardware properties of ibmq_mumbai on march 9th 2023 when the data acquisition of Figure 3.2a was done.

Figure 3.2: Scaling factor analysis

The syndrome rounds are decoded with the Bravyi-Haah decoder and the results are corrected according to those outcomes. The logical error probability P_e is calculated as described in Section 2.9 in Equation 2.14. The results, meaning the logical error probability P_e for each code distance d and round T are plotted as points in the graph. Each distance d is fitted according to the Equation 2.14 to derive T_s, the number of rounds T that fulfill the success criteria $P_s = 0.3$. T_s and ε_l are listed in tabulated form below the chart in Figure 3.2a.1). For both values, one standard deviation σ is given. Note regarding this fit: it seems probable, that there are also other error processes than just Pauli noise for small T. The points P_e for a certain distance for small T does not raise as fast as the fit suggests. Greenbaum and Dutton (2017) describe a model that takes into account coherent errors in the form of over rotations. After a certain number of QEC rounds T_{crit} the coherent errors become the dominant source of errors, while stochastic Pauli noise becomes less important. Usually, QEC Decoders account for stochastic Pauli noise. This might describe the observation at T<=10. When coherent errors take over it is difficult for the Decoder to detect the errors, and it results in a rapid increase of the logical error probability P_e. The mentioned model though, is limited to coherent noise induced by over rotations. It does not account for leakage, readout errors, and other unwanted qubit-qubit interactions. Nevertheless, this model is implemented as a potential variant and discussed in Section 3.8 [31].

In the next subgraph of Figure 3.2a.2) the points T_s are fitted with Equation 2.18, to derive ρ, the lifetime scaling factor. Values between 0 and 1 are possible for ρ, whereas the closer ρ is to 0, the better the lifetime scales with the code distance d. T_0 is the offset and the extrapolation of the curve to a distance of 0. This can be understood as the average lifetime T of the data qubits used by the ARC, at a distance of $d = 0$. For the run of ibmq_mumbai, the red trace (uncorrected error rate of data qubits) from Figure 3.1 at $P_e = 0.3$ and T_o fall together. Note, that T_o is vastly dependent on the parameters and window of the analysis, and sbecause it is an extrapolation, this is not always accurate.

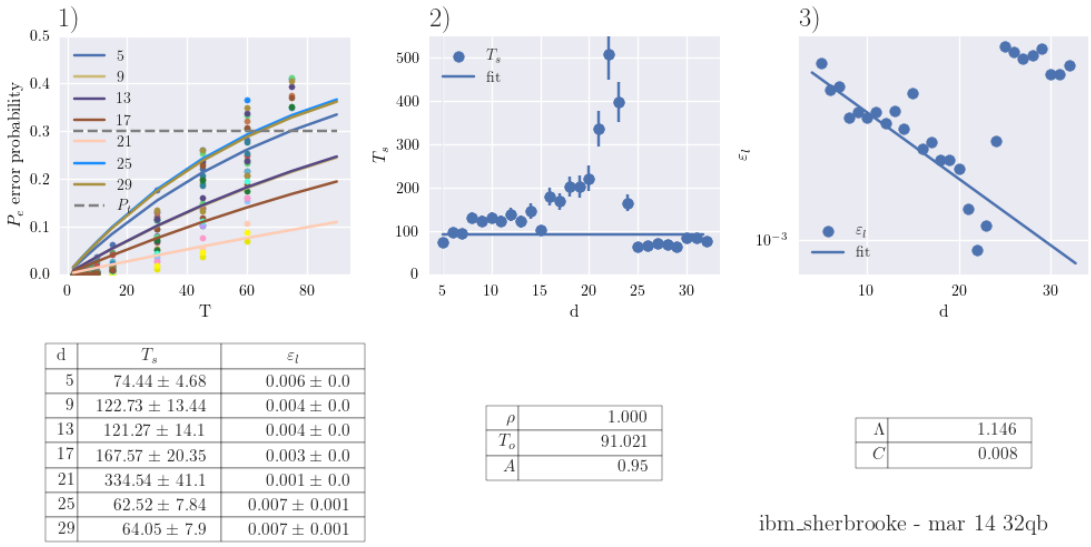
In Figure 3.2b a qualitative visualization of the hardware properties of ibmq_mumbai at the time of the circuit, execution is shown. The first two rows contain the error frequencies of the thermal relaxation times ($1/T_1$, $1/T_2$ in μs). Since T_1 and T_2 are the average times until a qubit suffers from a bit flip or a phase flip, a higher number indicates a more stable qubit in therms of information lifetime. Since all other values are error rates, in which the qubit performs generally better with lower values, the relaxation times were transformed into a relaxation frequency. In this sense, low numbers are favourable (blue) and high numbers are bad (red) and each column represents one qubit. Therefore, the qubits which have outliers in their error rates compared to the others can efficiently be detected. The qubits used in the linear chain of the ARC are indicated in the bottom row of this chart so that the reader can relate, to which qubits (columns) were used for which distance d. More work can be done to evaluate the influence of the number of shots per circuit execution and how this can reduce the spread of the data.

3.2 Large repetition codes

This section discusses the results from the ARC executed on larger quantum computers with over 100 qubits. ibm_sherbrooke is from the Eagle r3 processor type and contains 127 qubits, while ibm_seattle is from the Osprey r1 processor type with 433 qubits.

The ARC circuit execution was attempted on ibm_sherbrooke from distances $d = 5$ to $d = 40$, with up to 200 repetitions T per distance. The parameters were explicitly set to be far beyond the expectations of what this processor can execute, to learn more about the capabilities and limitations of such large quantum computers. It was possible, to obtain results for distances up to $d = 35$ which is remarkable, and repetitions up to $T = 60$. Runs beyond these parameters exceeded the capabilities of the current control hardware and failed for different reasons hinting towards control hardware limitations.

Figure 3.3 shows the visualization of the scaling analysis to deduce ρ and Λ as described in Section 3.1. Figure 3.3.1) shows the error probability points. Note, for visualization reasons only every fourth trace is displayed. Nevertheless, the fits were carried out with the full dataset. The data points display an interesting characteristic in Figure 3.3.2). The lifetime scales exponentially up to a distance of $d = 22$ and after that, T_s breaks down and remains relatively low for the rest of the measured distances. Since the model in equation 2.18 does not account for such breakdowns, the result for ρ is detrimental when analyzed of ther the full distance. The fit for Λ in Figure 3.3.3) does not seem to be much affected by this.



d	T_s	ε_l
5	74.44 ± 4.68	0.006 ± 0.0
9	122.73 ± 13.44	0.004 ± 0.0
13	121.27 ± 14.1	0.004 ± 0.0
17	167.57 ± 20.35	0.003 ± 0.0
21	334.54 ± 41.1	0.001 ± 0.0
25	62.52 ± 7.84	0.007 ± 0.001
29	64.05 ± 7.9	0.007 ± 0.001

ρ	1.000
T_o	91.021
A	0.95

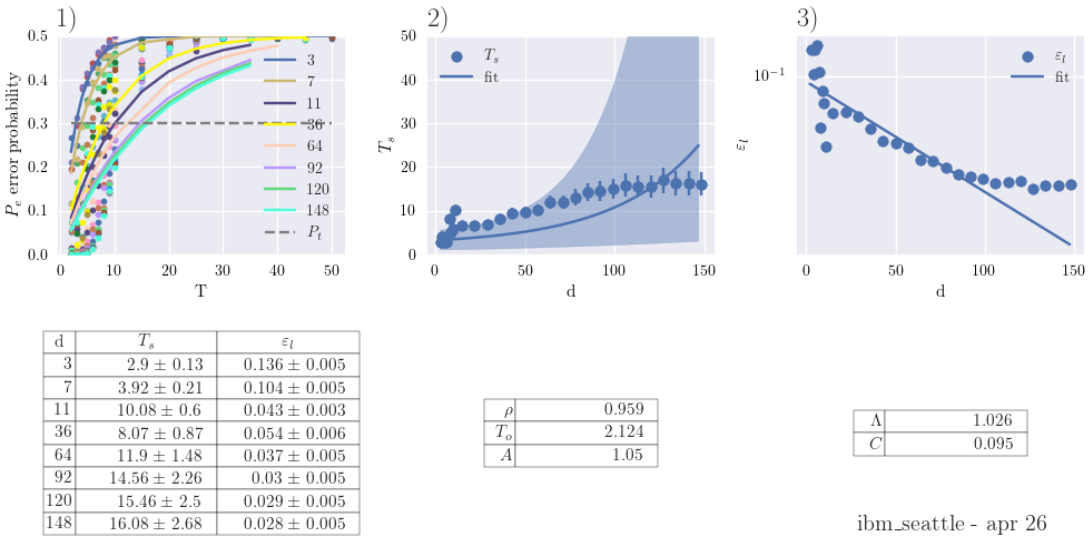
Λ	1.146
C	0.008

ibm_sherbrooke - mar 14 32qb

Figure 3.3: Standard scaling analysis for ibm_sherbrooke. 2) shows a logical error probability brake down after $d = 22$. 3) shows the same break down in form of higher error probabilities per round after $d = 22$.

A similar effect was found with ibm_seattle, during the test to run the ARC up to a

distance of $d = 148$ as shown in Figure 3.4.2). After $d = 10$, the logical error probability T_s dropped, and with increasing d another slower increase in T_s is observed, and after $d \approx 100$, it slowed down even further. In the logical error rate per cycle analysis in Figure 3.4.3) it shows clearly 3 different Λ values for different ranges of d. The first range is up to $d = 10$, the second range of Λ ranges from $d = 10$ to around 100, and after that it slowed down even further. Since the second range is the longest, the fit for Λ takes this value of $\Lambda = 1.026$. Compared to other quantum computers this is a low value. It is challenging to interpret this data. There are indications that there is a distance-dependent exponential noise process at work, which counters the gains in lifetime through the large repetition code.



d	T_s	ε_l
3	2.9 ± 0.13	0.136 ± 0.005
7	3.92 ± 0.21	0.104 ± 0.005
11	10.08 ± 0.6	0.043 ± 0.003
36	8.07 ± 0.87	0.054 ± 0.006
64	11.9 ± 1.48	0.037 ± 0.005
92	14.56 ± 2.26	0.03 ± 0.005
120	15.46 ± 2.5	0.029 ± 0.005
148	16.08 ± 2.68	0.028 ± 0.005

ρ	0.959
T_o	2.124
A	1.05

Λ	1.026
C	0.095

Figure 3.4: Scaling analysis of ibm_seattle. As with ibm_sherbrooke, the scaling behavior in 2) is only exponential for small distances and loses this property for larger distances. This is also visible in the Λ factor analysis in 3). The points indicate multiple Λ factors for different sections of d. For $d < 10$, a high Λ factor can be deduced, then a less high Λ is visible between $d = 10$ and $d = 100$ and even a lower value for the remainder of d. The analysis though, takes every point into consideration and therefore gives an average value for the Λ factor.

It is worth mentioning, that runs on ibm_seattle up until $d = 148$ with $T = 35$ cycles were executed successfully. It was attempted to decode the full dataset the Bravyi-Haah decoder, but after a run of $d = 56$ and $T = 35$ took 55 hours, this analysis was stopped. The more efficient PyMatching decoder finished the task in 1.1 hours. This topic is discussed in more detail in Section 3.4.

Runs on smaller quantum computers with 27 qubits sometimes also experience a drop in P_e after a distance close to $d \approx 10$. This can be caused by enhanced noise due to closing the loop in the linear ARC layout (only for the Falcon processors), or especially

error-prone data qubits that are added to the code, which reduce the overall performance of the ARC. Another explanation is that the decoder does not work well with closed-loop linear codes. Due to the limited distance of those ARC runs of Falcon devices, it is challenging to determine if there are just boundary effects, bad qubits or challenges for the decoder.

With the possibility to run large distance ARCs on numbers of qubits which are almost impractical for classical processing and decoding, there are indications for local maxima at certain distances at which T_s peaks. The analysis of the lifetime demonstrates, that the lifetime at a certain distance is the biggest, and the amount of errors per cycle is the lowest. On ibm_seattle, a local maximum arises around $d = 10$, and another around $d = 130$. But the differences in logical error probability are so minimal, that this interpretation has limited weight.

The source of this drop in performance is yet unknown and no literature reports similar effects or hints towards an explanation of such effects. One possibility is, that a certain amount of good qubits in a row at the beginning of the chain causes an exponential drop in logical error rate ε_l for $d < 10$, and all the following qubits are not as good anymore. Heating up of the environment due to the large overall amount of gates and mid-circuit measurements executed in this circuit could be another explanation. A third explanation could be, that other peaks are just not visible, because for larger distances not every single possible d was tested, but every 4th (too low sampling rate, so to say).

There is room for further studies on this topic, to verify the existances of an optimal distance in terms of lifetime, which does not occur at the biggest possible distance on a layout. If this optimal distance could be verified, the source of this drop in performance could be investigated.

To compare the scaling factors with other quantum computers, it is advised to restrict the distances down to a range in which they perform better. Selecting the optimal d for the large quantum computers by assessing the largest lifetime, or lowest error rate per cycle, yields more competitive results. Those are discussed in Section 3.7 in a comparison of many quantum computers.

3.3 Variability of quantum computers over time

Over the course of this project, changes in the properties of the quantum computers over time were noticed. This is not a new topic and is well explained by fluctuating external influences and different times past since the last calibration sequence. The impact on the performance of quantum error correction and therefore the ARC circuit can be noticed in the results. To examine the impact of the variations, different runs on the quantum computers with equal parameters executed on different time points were analyzed.

The quantum computers ibmq_mumbai, ibmq_montreal, and ibm_hanoi were compared in Figure 3.5 and it was found that the difference in scaling factors over a period of one and a half months were low. One qubit on ibmq_montreal was noticed to go out of order permanently, between the first and the second run between January and February. While the effect on the scaling factors was observable, it is interesting to see that the

Figure 3.5: Visualization of the ARC circuit scaling factors ρ and Λ of different quantum computers over different times points over a period between one to two months. The performance changes over time were low. On ibmq_montreal, one qubit went out of order permanently, which was noticable in the results.

ARC was still able to handle this gap in the qubit chain relatively well.

3.4 Comparison of the decoders Bravyi-Haah and PyMatching

Two different decoders were used for the analysis of the syndrome data. The first is the Bravyi-Haah decoder described in [22]. The second one is the PyMatching decoder, a minimum weight perfect matching code-based implementation in C++, with the goal of being extremely performant [21]. The difference in decoding quality and decoding runtime is already visible for small distances or repetitions T and develops linearly.

Figure 3.6 shows a comparison of the Bravyi-Haah decoder versus the PyMatching decoder on the truncated ibmq_kolkata datasets of distances $d = 6$, whereas decoding runtime is given in seconds on the left y-axis, the logical error probability P_e is given on the right y-axis. The performance in terms of low logical error rates differs roughly by a factor of two on this particular data set. Comparisons of other datasets yielded a wide diversity of differing performances in decoding quality. One trend appears: On truncated data sets and data sets where resets were used, Bravyi-Haah seems to outperform the PyMatching decoder, like the one presented in Figure 3.6. In terms of the decoding runtime, the PyMathing decoder is consistently faster, approximately 7-8 times.

Figure 3.7 shows the difference of the two decoders for distances above $d = 55$. Note, that the decoding process with the Bravyi-Haah decoders was stopped, after noticing the long runtimes for big distances, like $d = 56$ and $T = 35$. Therefore, there is a gap in the data traces. With the PyMatching decoder, the full dataset was decoded. The difference in runtime is striking. PyMatching is roughly 50 times faster, while maintaining a better

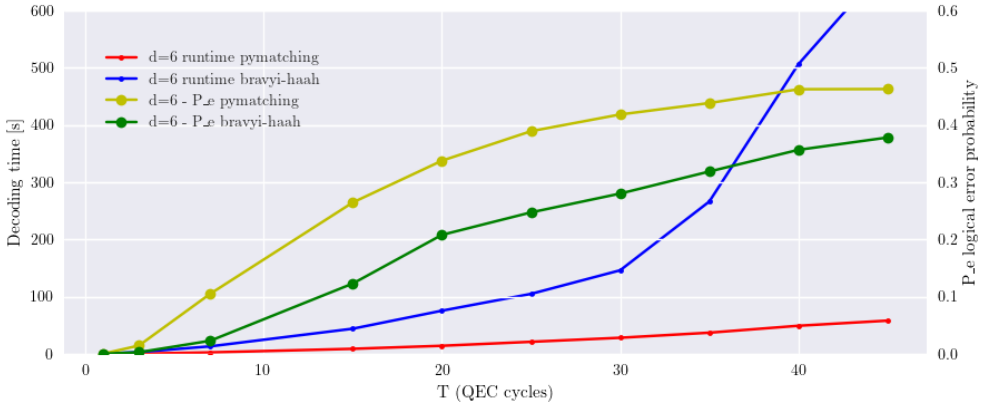


Figure 3.6: The left y-axis shows the decoding time in seconds, and the right y-axis the logical error rate P_e. The runtime and the decoding quality of the Bravyi-Haah and the PyMatching decoder are compared. PyMatching is 7-8 times faster, while the decoding quality varies for each dataset. For truncated dataset, there seems to be a trend towards the Bravyi-Haah decoder being better.

decoding quality.

The PyMatching decoder is faster in decoding large codes because the calculation of the edge weights for the decoding graph is done once in the preprocessing, before the decoding process itself is taken over by the PyMatching C++ library, which increases the efficiency. The decoding quality makes no major difference for large T, but before it starts to converge towards $P_e = 0.5$, PyMatching is better by 20-50%.

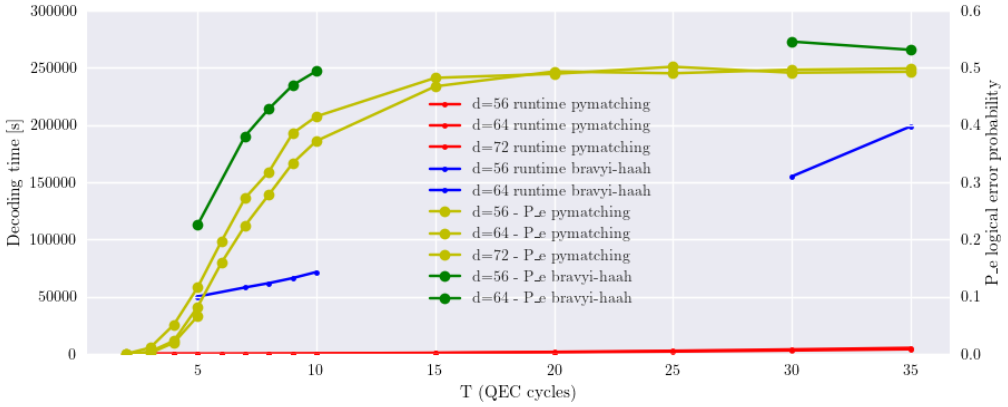


Figure 3.7: For larger codes, beyond $d = 10$, and beyond $d = 50$ the PyMatching decoder outperforms Bravyi-Haah in every considered aspect. The runtime is orders of magnitude better. While the Bravyi-Haah decoder needs over 55 hours to decode, PyMatching is completes the task in one hour.

A further advantage of the current PyMatching implementation is, that the edge weights of the decoding graph can be adjusted according to the different error rates and types per qubit. The PyMatching instance employed in this project uses optimized edge weights, derived by the single-qubit error rate calculation described in Section 2.6. Another advantage is that the PyMatching decoder can be adjusted to account for leakage (more deeply discussed in Section 2.5). This is already a commonly applied method in the QEC research [17, 18, 29, 32]. In this project, this trajectory was not further explored, since it is a specific optimization which is not straightforward with the given Qiskit framework and might not even be allowed by the new benchmarking rules (see Section 1.2 and [13]).

3.5 The influence of the basis

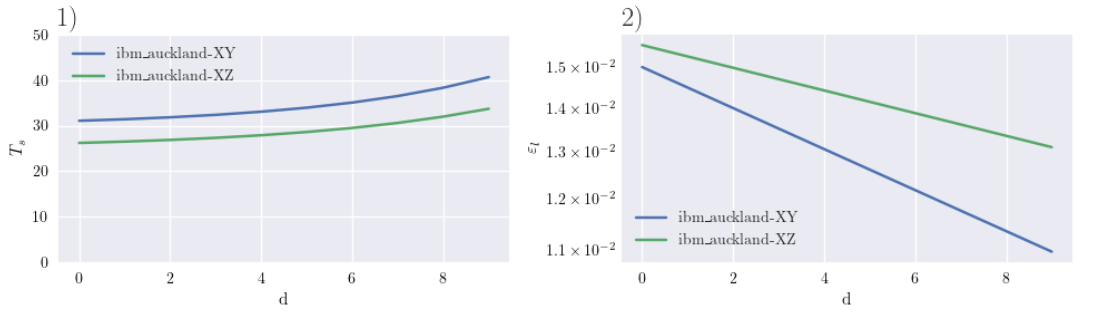
Two different basis pairs were used to encode the ARC. The default version is the XY basis, meaning that most of the datasets were acquired with this basis. The other basis is the XZ basis pair. If an error occured in the Z basis, it was almost certainly a bit flip, while an error in the X basis is a phase flip. For a flip in the Y base, it is impossible to deduce, whether it was a bit or a phase flip. Therefore, the advantage of using the XZ basis is that for the single qubit error analysis (Microbenchmark) the bit- and the phase flip rate can be deduced with high confidence, while in the XY basis, this is not possible. On the other side, the data qubits in the Y basis are vulnerable to bit and phase flips at the same time, which is favourable for error detection but disadvantageous for correction.

The IBM quantum computers measure all natively on the Z basis. Fewer gates need to be applied when using the XZ basis because the data qubits are encoded in Z don't need to be rotated before the measurement.

Before an ARC circuit can be executed on a quantum computer, it needs to be transpiled such that only the supported native gates of the targeted quantum computer are used. The relevant native gates of ibmq_mumbai for example are X, $\sqrt{x}$, and $R(Z)$. To initialize a data qubit in the X basis, the sequence $Rz(pi/2)$, $\sqrt{x}$, $Rz(pi/2)$ has to be applied. For the Y basis it is $Rz(pi/2)$, $\sqrt{x}$, $Rz(pi/2)$, $Rz(pi/2)$. So they differ in the number of $Rz(pi/2)$ gates applied to the qubit. Executing a $Rz(pi/2)$ takes roughly 10 ns each. Note, the available native gates and the circuit compilation results for the different quantum computers and gate execution times may vary. On some quantum computers, instead of a CNOT gate the ECR gate is used, which is executed in the X basis. Nevertheless, the difference between using another basis ends up in applying a different amount of single-qubit rotations.

Given, that the current IBM quantum machines have thermal relaxation times of around $100\mu s$ in average, the pure ARC execution time difference of altered basis pairs does not play a significant role. Therefore, it is assumed that there are no significant differences, due to the impact of a slightly longer execution times.

To evaluate the differences of the scaling factors, Figure 3.8 shows two different runs with varying basis pairs from ibm_auckland. The scaling factors are almost identical.



backend name	run	ρ	T_o	A
ibm_auckland	XY	0.615	29.865	0.95
ibm_auckland	XZ	0.642	25.015	0.95

backend name	run	Λ	C
ibm_auckland	XY	1.072	0.016
ibm_auckland	XZ	1.039	0.016

Figure 3.8: Comparison of the fits for ρ in 1) and fits for Λ in 2) for the basis pairs XY and XZ of ibm_auckland

The only difference is the value of T_0 in the lifetime scaling, where it seems that XY has a constant higher lifetime. The same trend is visible on ibmq_montreal in Figure 3.10, in which ρ and Λ are very close to each other, but T_0 in the XY basis is higher.

From this results on ibm_auckland, it is clear that the change in scaling is minimal, whereas XZ is roughly 5% worse than XY, and 10% worse in T_0, the constant offset in the lifetime. The difference in the Λ factor is so small, that it is negligible.

Comparing the physical decay rate of the single qubits involved under ARC execution yields, the majority of qubits decay faster using the XZ basis pair. This points towards a physical explanation of the difference. Comparing the single-qubit T1 and T2 times (from characterization) yields in all cases a better T1 time than T2. T1 times are associated to bit flips which can occur in the Z and the Y basis, T2 times are associated to phase flips which can occur in the X and the Y basis. There is no simple explanation for why the ARC execution produces more errors when using the Z basis, instead of the X basis, under the condition that the qubits have a better T1 time. There must be a factor, which makes the ARC expecially sensitive to bit flips.

Analyzing the data from ibm_auckland in more detail shows a significant difference. The P_e of the XZ dataset for small T and $d = 10$ starts at 1, and goes down below $P_e = 0.5$ around $T = 15$. The leakage analysis yields tremendous leakage such that the decoding algorithm requires to deal with ancillas stuck in the 1 state. The XY dataset, on the other hand, shows a strong T-dependent even-odd effect of P_e. For $T = 1$ there is $P_e = 0.95$, for $T = 2$, there is $P_e = 0.05$, and this zig-zag goes on until $T = 20$ in which it levels off around $P_e = 0.3$ as visualized in Figure 3.9. Both datasets indicate strong leakage, but the leakage seems to affect the Z basis stronger. The Y basis is partially affected by leakage, depending on even or odd T. The T even-odd dependency may come from the fact, that no resets are not used and the ancilla qubit flips with every round T between 0 and 1. It is known, that leakage affects other qubits only if they are in the $|1\rangle$ state [18]. More research can be carried out to investigate the possibility that leakage

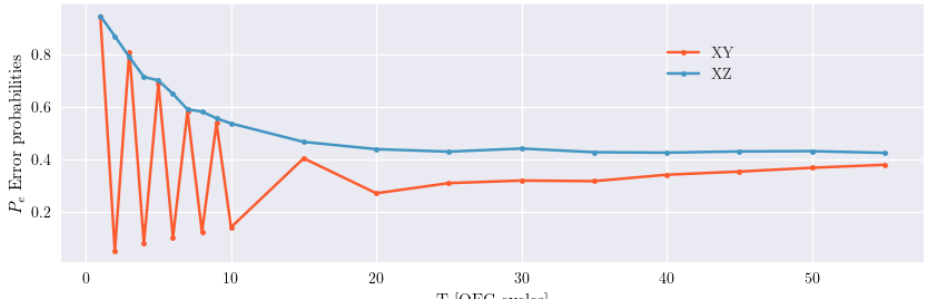
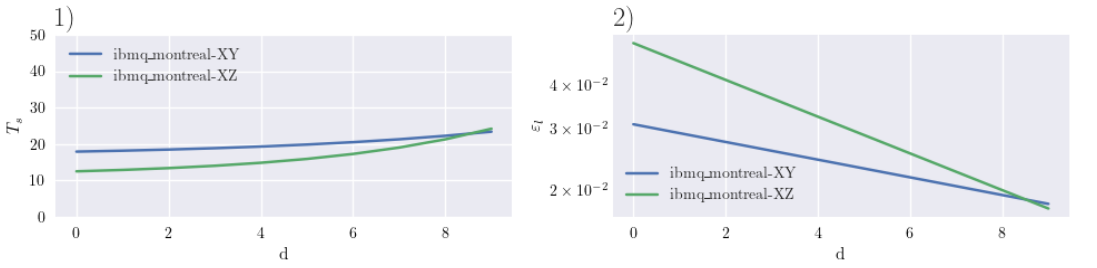


Figure 3.9: Visualization of two data traces of $d = 10$ for XY and XZ basis pair of ibm_auckland. The trace for XY shows a even-odd T dependent logical error rate and the XZ trace starts with a $P_e = 0.95$. For both traces, is an indication of leakage and the leakage occurance is basis dependent.

happens on the Y-basis, and therefore a Y-based 0 state might not be affected. ECR gates sometimes replace CNOT gates (depending on available native gates and circuit compilation) and the ECR gate maximally entangles the qubits in the X basis. This could explain why the Z basis is maximally affected by leakage, but Y basis encoded qubits only partially. These are all rather speculations than final conclustions and the problem remains unsolved. It is likely that leakage is the key to an explanation of this phenomena and more research needs to be conducted to find certainty.



backend name	run	ρ	T_o	A
ibmq_montreal	XY	0.689	16.649	1.05
ibmq_montreal	XZ	0.604	11.146	1.05

backend name	run	Λ	C
ibmq_montreal	XY	1.124	0.032
ibmq_montreal	XZ	1.274	0.059

Figure 3.10: Scaling factor comparison for ρ in 1) and for Λ in 2) of using different basis pairs on ibmq_montreal.

Figure 3.10 shows the same analysis as before for ibmq_montreal, but the scaling values are better for the XZ basis than for XY. The main difference in circuits executed on ibmq_montreal and ibm_auckland is, that ibm_auckland uses the ECR-gate instead of the CNOT gate.

3.6 Using resets after mid-circuit measurements

Resets are an important topic in QEC. On the one hand, resets mitigate the propagation of leakage which is challenging for common decoders. On the other side, resets are a heavy operation for the QC controller hardware and limit the number of possible QEC cycles. During this work, ARC runs with resets were not executed successfully beyond $T = 15$, whereas without resets T's beyond 150 were observed. Nevertheless, the quality of the results drops at large T.

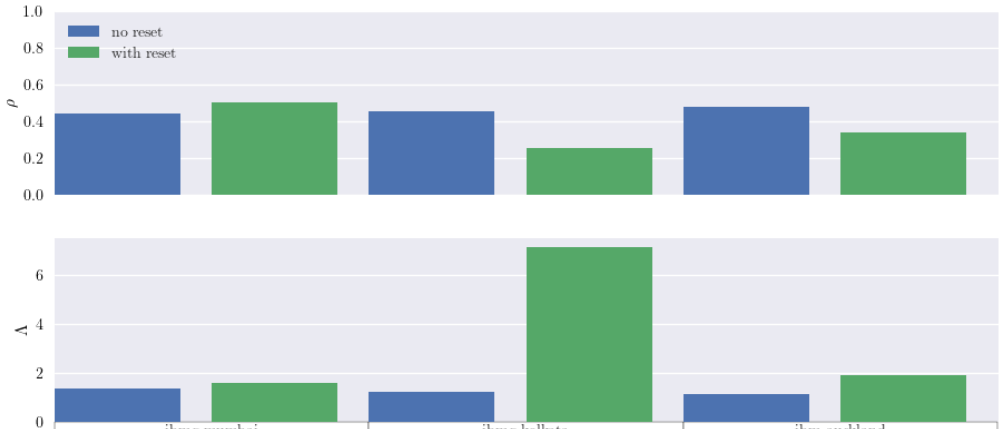
Figure 3.11 shows a comparison of different quantum computers with and without resets, in two charts. The upper chart compares ρ where as the lower chart compares the Λ factor. Remember, that for ρ smaller values indicate better scaling in lifetime and for the Λ factor a higher value indicates a better scaling behaviour error rate per cycle ε_l.

In the case of ibmq_mumbai, runs with resets reached $T = 10$. The logical error rate stayed below 0.2, even for the smallest distance. Compared to runs the same parameters in terms of d, and T but without resets, the results are in the same range. This is an unexpected and remarkable finding. For ibmq_mumbai, the difference in lifetime scaling ρ is 14 % where runs with resets performed worse, and in the error rate scaling factor Λ is 17 % worse.

Similar results were found on ibmq_kolkata, but the other way around. T was limited to 6 for the analysis of both datasets, with and without resets, because runs with resets and $d > 6$ did not run successfully due to hardware constraints. This limitation of T yields remarkably good results in the dataset without resets. Runs on ibm_kolkata without resets produce results within 77% to the run with resets. Surprisingly, Λ with resets is almost 500% higher than without resets. Due to the very sparse dataset of ibmq_kolkata on the run without resets, those results have limited relevance.

On ibm_auckland, the situation is slightly different. Datasets without resets are only available for runs with $d = 10$, and smaller distances were produced through truncation (see Section 2.12). For the dataset with resets, the full spectrum of distances is available. Therefore the comparison happens under not exactly matching circumstances. Here, the dataset with resets clearly outperforms the other dataset by 40 % for the lifetime, and 60 % for the Λ factor. From the analysis of the basis, it is known that ibm_auckland is prone to leakage. A studie about leakage has shown that resets have a strong influence to mitigate the further spread of leakage [18]. Therefore, the result is in line with the expectation that resets are helpful to mitigate leakage. To further improve the relevance of this result, equal circumstances must be used by creating a truncated dataset from the run with resets.

The impact of the reset after each syndrome readout seems to be limited in terms of the scaling factors ρ and Λ when operating in small numbers of repetition cycles. An explanation for the worse scaling with ibmq_mumbai might be, that resets take time. The total execution time of the circuit is longer, which allows for thermal relaxation or Pauli errors to happen on the data qubits while the ancillas are being reset. But information lifetime measured in seconds could yield an advantage because the circuit execution duration with resets takes approximately 30% more time. An explanation,



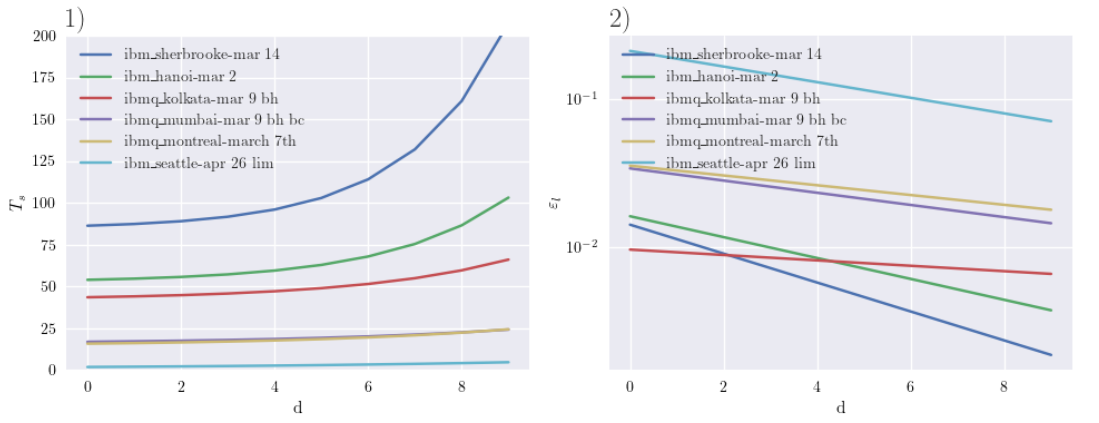
	ibmq_mumbai	ibmq_kolkata	ibm_auckland
no reset ρ	0.443	0.456	0.478
with reset ρ	0.506	0.257	0.341
diff ρ	0.064 (14.4)%	0.199 (77.5)%	0.137 (40.2)%
no reset Λ	1.365	1.208	1.161
with reset Λ	1.592	7.115	1.919
diff Λ	0.227 (16.6)%	5.907 (488.9)%	0.758 (65.3)%

Figure 3.11: Barchart displays a comparison between runs with- and without resets for the quantum computers ibmq_mumbai, ibmq_kolkata and ibm_auckland. The upper chart compares ρ and the lower chart Λ, in which the bars with no resets are blue and with resets are green.

why the scaling of Λ and ρ drops significantly for more cycles in the case of no resets, might partially be found in coherent errors caused by over rotations, as described in a paper from Greenbaum and Dutton (2017) [31]. The main concept is that very small over rotations in gates accumulate over time (or cycles) and cause non-Pauli errors. Hence in a small number of cycles, Pauli errors dominate and after a certain point, coherent errors take over. Since most of the decoders are developed and tested to handle Pauli errors, they start to fail more frequent and the logical error rate increases beyond Pauli error-based expectations.

3.7 Rank ordering Quantum Computers with the QEC Benchmarking

The goal of a benchmarking framework is to compare devices based on defined criteria. In this case, it is the ability of a quantum computer to perform quantum error correction, specifically on the property of how well it scales in lifetime ρ and error rate per QEC cycle Λ. In the following Figure 3.12 the most recent ARC scaling diagnostics runs's results of six different quantum computers are shown.



backend name	run	ρ	T_o	A
ibm_sherbrooke	mar 14	0.386	84.673	1.05
ibm_hanoi	mar 2	0.452	52.594	0.95
ibmq_kolkata	mar 9 bh	0.535	42.142	1.05
ibmq_mumbai	mar 9 bh bc	0.645	15.743	0.95
ibmq_montreal	march 7th	0.639	14.524	1.05
ibm_seattle	apr 26 lim	0.752	0.769	0.95

backend name	run	Λ	C
ibm_sherbrooke	mar 14	1.569	0.018
ibm_hanoi	mar 2	1.385	0.019
ibmq_kolkata	mar 9 bh	1.088	0.01
ibmq_mumbai	mar 9 bh bc	1.209	0.037
ibmq_montreal	march 7th	1.165	0.038
ibm_seattle	apr 26 lim	1.275	0.239

Figure 3.12: Rank ordering of the quantum computers based on the exponential scaling factors for the lifetime ρ (in 1)) and Λ for error rate per QEC cycle (in 2)).

Note, better values for ρ closer to 0, while for Λ higher values are desirable. Ordered by ρ and T_0 ibm_sherbrooke is leading by a good margin, followed by ibm_hanoi and ibmq_kolkata and ibmq_mumbai. ibmq_montreal is second to the last and by the time of writing this thesis already sunset.

The newest child in the family of IBM quantum computers is ibm_seattle, with 433 qubits. It was used to create the probably worlds biggest repetition code with a distance of $d = 148$ (as of May 2023). Just $d = 10$ is used to compare with the other smaller quantum computers with similar parameters, because of the scaling issues discussed in Section 3.2. ibm_seattle ranks last in terms of the scaling factor ρ, while it yields a competitive Λ scaling factor of errors per cycle. For the comparative analysis, $d = 10$ with open loop layout and $T <= 30$ was used for all devices.

Table 3.1 lists the results of the comparison for the quantum computers ordered by ρ. It is interesting to see, that the quantum volume (QV) is not a good predictor for the

QEC Benchmarking results											
Name	ρ	T_o	Λ	$T_1[\mu s]$	$T_2[\mu s]$	QV	Qbt	ECR	SXE	EOR	proc. type
ibm_sherbrooke	0.39	85	1.57	288	166	32	127	7.8e-3	2.3e-4	1.0e-2	Eagle r3
ibm_hanoi	0.45	53	1.39	183	128	64	27	-	2.1e-4	1.0e-2	Flc. r5.11
ibmq_kolkata	0.53	42	1.09	111	77	128	27	9.8e-3	2.5e-4	1.4e-2	Flc. r5.11
ibmq_mumbai	0.64	16	1.21	133	159	128	27	8.3e-3	2.1e-4	2.0e-2	Flc. r5.10
ibmq_montreal	0.64	15	1.17	93	88	128	27	-	-	-	Flc. r4
ibm_seattle	0.75	0.8	1.28	86	60	-	433	2.0e-2	6.0e-4	6.3e-2	Osprey r1

Table 3.1: Ranking of six IBM quantum computers based on its QEC diagnostics scaling values ρ and Λ. Mind, properties like ρ, Λ, QV, and thermal relaxation times and error rates are subject to constant change. The values depicted here are taken in mid-May 2023. According to the ρ value, ibm_sherbrooke is leading, followed by ibm_hanoi and ibmq_kolkata. Abbreviations used are number of qubit (Qbt), median echoed cross-resonance gate error (ECR) is equivalent to the two-qubit gate CNOT, median $\sqrt{x}$ error (SXE) is equivalent to single qubit gates, median error of the readout (EOR).

scaling of QEC codes. The two best-performing devices have a QV of 32 and 64 while the other three have a QV of 128 and ibm_seattle is not rated yet.

The most obvious metric that comes in almost the same order as the ρ value, is the readout error followed by the thermal relaxation times T1 and T2. Noteworthy is, that for ibmq_mumbai the thermal relaxation times are competitive, the only parameter that can explain the bad ranking is the higher readout error. This makes sense, because in QEC readout, especially mid-circuit readouts is a very often used gate (compared to other algorithms).

Previous studies on more complex QEC codes on a subset of the here-used devices were conducted by Wotton et al. (2022). [37]. Since not the same quantities were measured, values can not be compared directly. Nevertheless, error rates were measured for each quantum computer and according to this, the same ranking of the quantum computers was observed. ibm_hanoi was better than ibmq_kolkata, which was better than ibmq_montreal.

Google Quantum AI et al. (2021) studied standard repetition codes up to a distance of $d = 11$ on the Sycamore quantum processor. They achieved $\Lambda \approx 3$ [32]. While the same quantity was measured, it needs to be pointed out, that different optimization techniques were applied. Furthermore, bit flip and phase flip repetition codes were used, which are assumed to be less noise sensitive. They used the truncation technique, to derive datasets of smaller distances from runs of $d = 11$, but the smaller distances were averaged over the data qubits of the full distance. As will be discussed in Section 3.9, this technique alters the data, and is not always in good agreement with the actually measured data.

3.8 Using a model that accounts for coherent errors

As previously mentioned in Section 3.6, the model that only considers stochastic Pauli noise, only suboptimally fits the P_e points for small T. Greenbaum and Dutton (2017) propose a model that accounts for coherent over rotations, which is introduced in Section

2.9.2 [31]. Subsequently the results of the modified scaling analysis are presented, which uses an additional term to account for coherent noise induced by over rotations. It is assumed to be a dominant source of noise for large T.

Figure 3.13a shows the scaling analysis. The window of analysis was constrained to T between 2 and 30 because, unlike the first term of stochastic Pauli noise it does not converge to $P_e = 0.5$, and this leads to more inaccuracies when analyzing the full T. For the comparison with the basic model, the same analysis with only the stochastic Pauli noise term is shown in Figure 3.13b, with the same constraints applied to T.

The fit of the modified model follows more accurately the first few data points at low T, compared to the analysis with the uncorrected model. This closer fit leads to a significant change in the scaling factor of the lifetime ρ. Unfortunately, it is more difficult to estimate the spread of T_s, since this value is no longer obtained analytically due to the more complex model. It was sampled by applying the model to a series of T values and the closest value to P_s was chosen. The scaling factor of the logical error on the other hand increased by a large margin from $\Lambda = 1.2$ to $\Lambda = 1.79$. The σ value of ε_l on the other hand allows to be compared to the simple model. It turnes out, that σ was reduced by more than a factor of two.

Overall, such a refinement of the model is a step in the right direction, but the modeling for leakage is still missing. More refined models for other error processes should be applicable for the full range of T, which obviously saturates around $P_e = 0.5$.

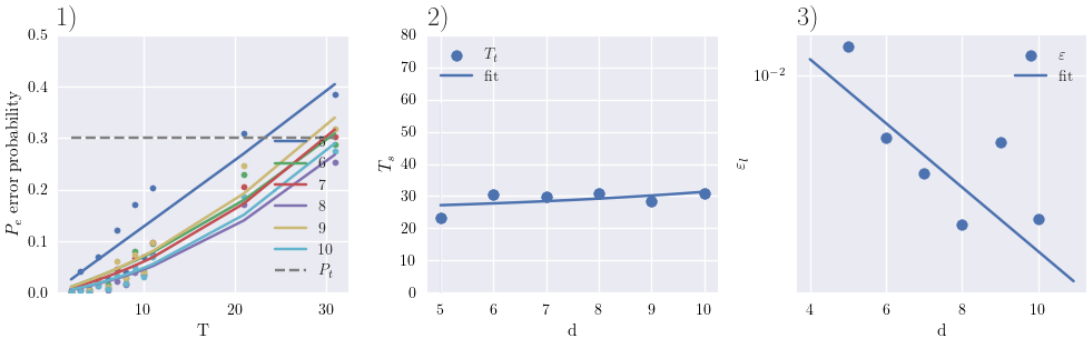
3.9 Deriving datasets with truncation

Truncation is a useful technique to derive sufficient syndrome data for analyses, but also limits the interaction with the quantum computer. The technique applied to derive the data points is described in the Theory Section 2.12. Figure 3.14a presents a comparison of the full dataset acquired from ibm_sherbrooke. The logical error probability P_e is plotted over d for several selected T. In red, the data from the full data set is shown, while in blue the derived data points are shown, which stem from a single $d = 22$ circuit execution, for several T. From this $d = 22$ data row, the truncated lower distance data rows are created by subsequently cutting away data qubits and ancillas from the syndrome data. The newly created derived syndrome dataset is again decoded.

When considered from the right side of the graph, at $d = 22$, both, the corresponding blue and the red traces start at the same point. Both points stem from the same syndrome dataset. While the red trace shows points for smaller d from real quantum computers, the blue ones are derived from the connection point to the right. The trajectory of follows roughly the same direction, but the truncated traces are more smooth and average out sharp turns in the trace of the real quantum computers. At a distance of 4 (or 3 for truncated traces), a difference in P_e is visible.

To witness the differences in the dataset from another point of view, and to evaluate how this impacts the scaling analysis, the standard analysis Figure is shown for the truncated data set in Figure 3.14b.

The truncation provides an interesting picture, with a smooth increase in lifetime T_s



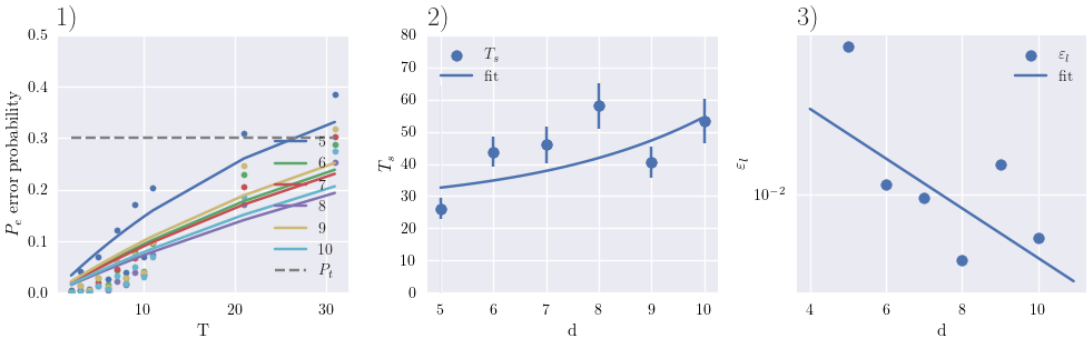
d	Tp_t	ε_l	$\bar{\varepsilon}$
5	23.29 ± 0.01	0.013 ± 0.003	0.0323
6	30.37 ± 0.01	0.006 ± 0.001	0.0365
7	29.98 ± 0.01	0.004 ± 0.001	0.0411
8	30.99 ± 0.01	0.003 ± 0.001	0.0401
9	28.53 ± 0.01	0.005 ± 0.002	0.0403
10	30.99 ± 0.01	0.003 ± 0.001	0.042

ρ	0.706
T_o	24.206
A	1.05

Λ	1.793
C	0.05

ibmq_mumbai - jan 16 coh

a Scaling analysis of the advanced model that accounts for coherent errors. The fit in 1) follows visibly more closely the points, than the simple model in b.1)



d	T_s	ε_l
5	26.17 ± 3.14	0.017 ± 0.002
6	43.78 ± 4.81	0.01 ± 0.001
7	45.89 ± 5.68	0.01 ± 0.001
8	57.98 ± 7.12	0.008 ± 0.001
9	40.55 ± 4.94	0.011 ± 0.002
10	53.35 ± 6.81	0.009 ± 0.001

ρ	0.550
T_o	26.304
A	1.05

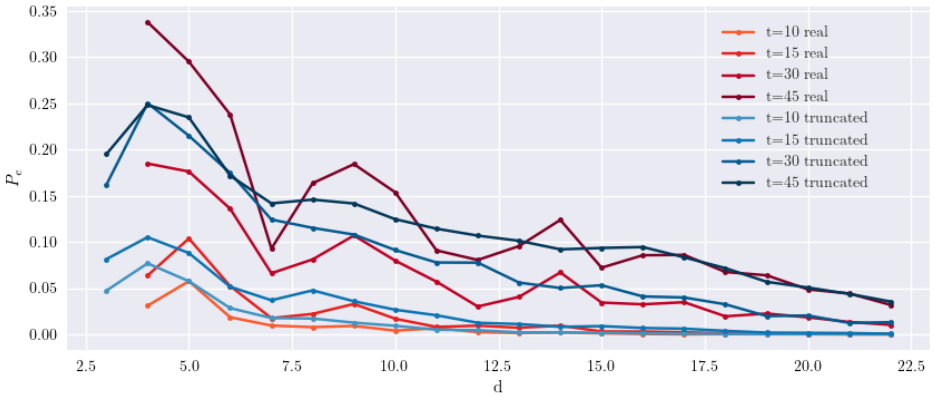
Λ	1.201
C	0.022

ibmq_mumbai - jan 16 coh comp

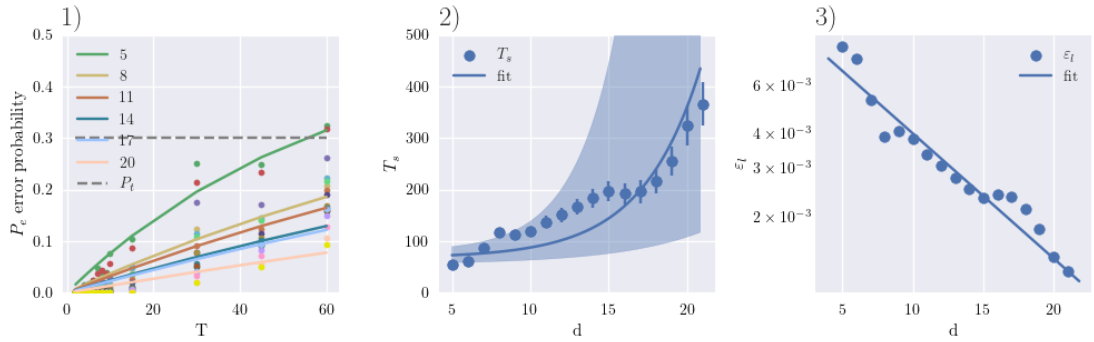
b Scaling analysis with the same parameters as a) but from the simple model.

Figure 3.13: a Comparison of the advanced model that accounts for coherent errors in the form of over-rotation and the standard model of stochastic Pauli errors in b.

and ε_l. Compared to many other datasets and scaling analyses, this presents itself as very clean and smoth, with points following the fits.



a Comparison between a full dataset and a truncated dataset on P_e over d. The red traces belong to datasets that were fully acquired from a quantum computer, while blue shows the truncated versions of those, based on the red biggest distance run.



d	T_s	ε_l
5	55.24 ± 3.53	0.008 ± 0.001
8	118.03 ± 8.46	0.004 ± 0.0
11	137.33 ± 11.66	0.003 ± 0.0
14	183.36 ± 18.08	0.002 ± 0.0
17	196.3 ± 22.18	0.002 ± 0.0
20	324.22 ± 38.66	0.001 ± 0.0

ρ	0.585
T_o	67.841
A	1.05

Λ	1.233
C	0.013

ibm_sherbrooke trk -

b Scaling analysis of the truncated dataset of ibm_sherbrooke.

Figure 3.14:

Figure 3.15 shows a comparison of many full datasets, with their truncated counterpart. There, also this current example of ibm_sherbrook is shown. The comparison of ρ shows a very minimal difference, below 1 %. The Λ factor differs with 10 % where as the truncated version performed better.

Other datasets from the Falcon 27 qubit devices were also truncated by using the closed loop $d = 10$ dataset (Note, open and closed loop layouts were introduced in Section 2.1).

The final results of the scaling parameters are provided in the following Figure 3.15 as a comparison.

	ibmq_montreal	ibmq_montreal	ibm_hanoi	ibmq_mumbai	ibmq_kolkata	ibm_sherbrooke full
full ρ	0.669	0.693	0.510	0.659	0.564	0.590
trk ρ	0.652	0.640	0.578	0.645	0.542	0.585
diff ρ	0.017 (2.6)%	0.053 (8.3)%	0.068 (13.3)%	0.014 (2.1)%	0.022 (4.1)%	0.005 (0.9)%
full Λ	1.116	1.089	1.241	1.138	1.062	1.103
trk Λ	1.150	1.164	1.198	1.087	1.490	1.233
diff Λ	0.034 (3.1)%	0.074 (6.8)%	0.044 (3.7)%	0.051 (4.7)%	0.429 (40.4)%	0.130 (11.8)%

Figure 3.15: Comparison of the difference between the full dataset and the truncated one for different quantum computers like ibmq_mumbai (for two runs of different dates), ibm_hanoi, ibmq_mumbai, ibmq_kolkata and ibm_sherbrooke.

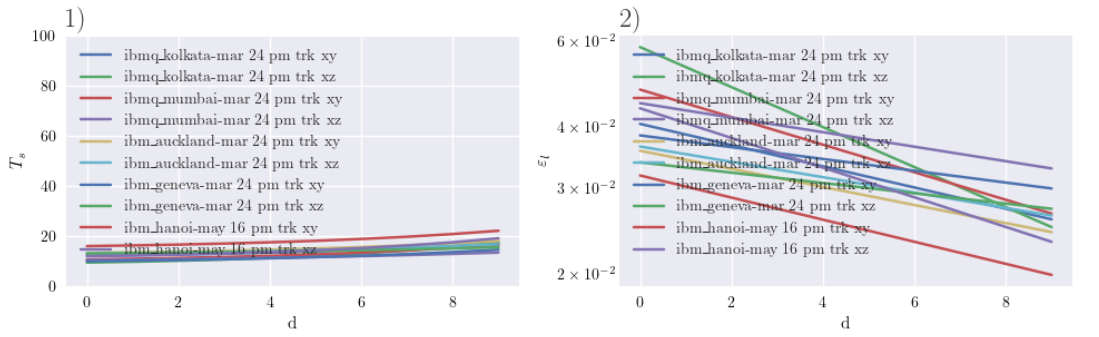
The differences of ρ vary, but stay below 5% on most of the compared data sets, and the same is true for the Λ factor. The only exception here is ibm_kolkata with a difference of 40 %. Obviously, truncation has a smoothing effect to data. So when the ε_l values are widely on the acquired dataset, truncation tends to remove this spread.

This approach of one-sided truncation seems to work well. Note, that it is important to truncate from the correct side, to reproduce the way in how the qubits were added to the chain when the full data set was acquired. Truncation from the wrong end, yields chaotic data.

The truncation method was tested in a large-scale comparison. Single-distance runs were executed on many quantum computers of the Falcon architecture. The parameters were kept as equal as possible on all the machines. The parameters are a distance of 10, a closed loop linear chain on each machine, and a fixt number of QEC cycles of $T = 30$. For every machine, both sets of basis pairs were recorded, XY and XZ, as well as both logical states 0 and 1.

Sherbrooke and Seattle do not compete in this category, because they use a different connectivity scheme, and were not recorded in this benchmarking batch with these configurations. Figure 3.16 shows the outcome of this benchmarking run. Two charts are shown for the two scaling factors ρ (to the left in Figure 3.16.1)) and Λ (right side in Fig-

ure 3.16)). The fit parameter values are listed below the chart. Abreviations in the run field are defined as follows: pm means that the PyMatching decoder was employed, trk means the dataset was derived from truncation and xy or xz indicate the used basis-pair.



backend name	run	ρ	T_o	A
ibmq_kolkata	mar 24 pm trk xy	0.700	10.883	1.05
ibmq_kolkata	mar 24 pm trk xz	0.642	8.157	1.05
ibmq_mumbai	mar 24 pm trk xy	0.683	9.449	1.05
ibmq_mumbai	mar 24 pm trk xz	0.757	9.252	1.05
ibm_auckland	mar 24 pm trk xy	0.704	12.088	1.05
ibm_auckland	mar 24 pm trk xz	0.732	11.781	1.05
ibm_geneva	mar 24 pm trk xy	0.697	8.84	0.95
ibm_geneva	mar 24 pm trk xz	0.777	11.857	1.05
ibm_hanoi	may 16 pm trk xy	0.677	14.76	1.05
ibm_hanoi	may 16 pm trk xz	0.659	10.722	1.05

backend name	run	Λ	C
ibmq_kolkata	mar 24 pm trk xy	1.105	0.042
ibmq_kolkata	mar 24 pm trk xz	1.208	0.064
ibmq_mumbai	mar 24 pm trk xy	1.139	0.051
ibmq_mumbai	mar 24 pm trk xz	1.071	0.046
ibm_auckland	mar 24 pm trk xy	1.089	0.037
ibm_auckland	mar 24 pm trk xz	1.075	0.038
ibm_geneva	mar 24 pm trk xy	1.057	0.039
ibm_geneva	mar 24 pm trk xz	1.05	0.034
ibm_hanoi	may 16 pm trk xy	1.11	0.033
ibm_hanoi	may 16 pm trk xz	1.15	0.047

Figure 3.16: The left part of the chart shows the fitted lines of ρ for each quantum computer and the right chart shows the fits for the Λ factor and the fit values are listed below the chart. Abreviations in the run field are as follows: pm means PyMatching, trk means truncation and xy or xz indicate the used basis-pair. All lines in the big comparison stem from closed-loop truncation-based ARC runs for different basis-pairs from ibmq_kolkata, ibmq_mumbai, ibm_auckland, and ibm_geneva. The result indicates that this method of truncation is not optimal for this kind of analysis, because the values are too similar to each other, and compared to the data acquisitions of the full parameters space, the scaling factors diverge apart.

The outcome of this test run of using one-sided truncation to reduce the number of executed circuits on the quantum computer, contains no adequate indicator of a clear ranking. The values of ρ and Λ are very close to each other and do not allow for a serious separation. Furthermore, compared to previous runs of fully acquired datasets or their truncated counterparts, the scaling values here are low. The problem was found in choosing the wrong decoder. As it turnes out, using the PyMatching decoder yields suboptimal results on truncated datasets.

Something remarkable appears when studying the comparison of these data sets. The T_s value of the $d = 10$ closed loop layout in the fully acquired data set was always lower than the one for $d = 10$ with open loop layout. This behavior was found in almost every data set which went up to a distance of 10. What makes this closed loop layout $d = 10$ so poor? Compared to the open loop layout, no more data qubits are included, only

an additional ancilla qubit to make syndrome measurements with the first and the last qubit of the chain.

Therefore, $d = 10$ does not add another code qubit, but solely more CNOT operations. As already mentioned in the supplementary information of a paper by Google Quantum AI (2021) [32], boundary qubits perform better than the ones in the middle (at least in repetition codes) because they are less involved in parity measurements. Therefore, the 10th data qubit in the open loop layout performs slightly better than qubits in the middle. Making this 10th qubit also a qubit 'in the middle' by closing the loop, while not adding another data qubit to store quantum information, affects the overall performance of the closed loop $d = 10$ rounds and yields a consistently worse outcome compared to the ones of the open loop. Another possibility is, that the decoders do not handle the closed loop well.

Certainly, it would be beneficial to make the comparison with the open loop layout, because on almost every machine this is the better configuration, which allows a clearer separation of the individual performances. For real and fair comparisons, the closed-loop layout problem might be mitigated, when random linear chains are employed, with random starting points. Such a possibility is discussed in Section 3.12.

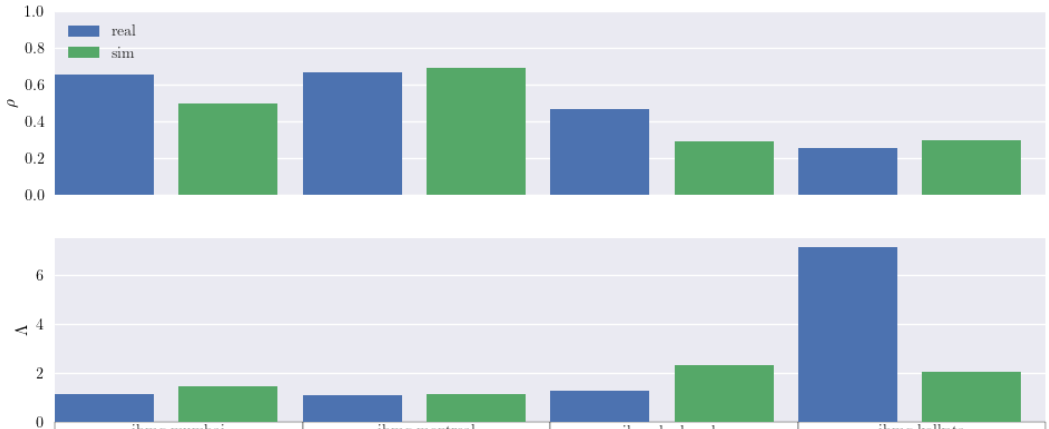
For the issue of using PyMatching for the decoding, it is recommended to test Bravyi-Haah as a decoder for truncation based data sets.

3.10 Comparison to simulations with property derived Pauli noise models

A further method to derive syndrome data is to use simulations. As described in Section 2.13, the noise models stem from the actual hardware properties of individual quantum computers, like single and multi-qubit gates, as well as the thermal relaxation times of the qubits. ARC circuits were simulated on the noise models of different quantum computers, with distances up to $d = 10$. The scaling analysis was performed on them. The outcomes of the simulations are displayed in Figure 3.17 in bar charts. The upper chart compares ρ, and the lower chart Λ. The fit parameter values and the differences are shown in tabulated form below. The results of the simulations varies significantly compared to their real quantum computer counterparts. For ibmq_mumbai, ρ and Λ are around 30% better in the simulation run. For ibmq_montreal the simulation is around 6% better and for ibm_sherbrooke the scaling values were 60% and 80% better. Ibmq_kolkata is the only exception in this trend of the simulations outperforming their real counterparts. On ibmq_kolkata, the simulation performed 16% worse in ρ and 240% worse in the Λ factor. Since this data set of ibmq_kolkata repetitively countered trends and contains only a small number data points, the reliability of this particular dataset is questionable.

As already discussed on multiple occasions, real quantum computers do not only suffer from Pauli noise. Coherent errors and leakage are further sources of errors. Therefore, it is not surprising, that the simulations overestimate the quantum computers.

Daniel Egger and Conrad Haupt (2023) developed a simulator for leakage [38, 39]. This might be a trajectory to further explore more accurate simulation techniques.



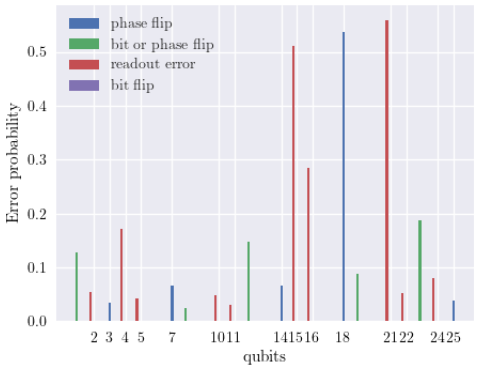
	ibmq_mumbai	ibmq_montreal	ibm_sherbrooke	ibmq_kolkata
real ρ	0.659	0.669	0.470	0.257
sim ρ	0.497	0.693	0.291	0.298
diff ρ	0.162 (32.5)%	0.024 (3.6)%	0.179 (61.3)%	0.041 (16.1)%
real Λ	1.138	1.116	1.258	7.115
sim Λ	1.466	1.136	2.311	2.066
diff Λ	0.327 (28.8)%	0.021 (1.9)%	1.053 (83.7)%	5.049 (244.4)%

Figure 3.17: The results of simulations (green) compared to the data acquired on their real quantum computer counterparts (blue) with similar parameters show significant differences in ρ (upper chart) and Λ (lower chart). The trend is, that simulations overestimate the performance of the real quantum computers.

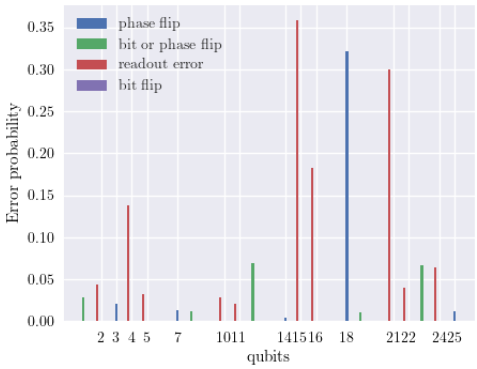
3.11 The Microbenchmark

The microbenchmark is a tool to reuse the already gathered syndrome measurements to assess the quality of each single qubit and derive error rates. This information might be interesting in the future of Fault-Tolerant Quantum Computing (FTQC) where QEC is applied for every circuit and the end users are only interacting with the logical qubits. Under the hood, syndrome measurements are probably still the main mechanism to detect and correct errors. Therefore, syndrome-based statistics is gathered and analyzed along normal circuit execution to obtain insight into qubit quality and drifts. This might eventually make calibration sequences obsolete. Along this thought train, it might even be possible to utilize this information to apply optimal control during arbitrary circuit execution. The pulse shapes are changed by infinitesimal amounts, while syndromes are analyzed in real-time, to assess if the changes were beneficial or not. A requirement for this is solid QEC such that risking to damage single qubits performance does not impact high-level circuit execution. A further more directly applicable use case for the micro benchmarks is to apply them in the decoder as weights for the decoding graph, to help prioritize the error sources.

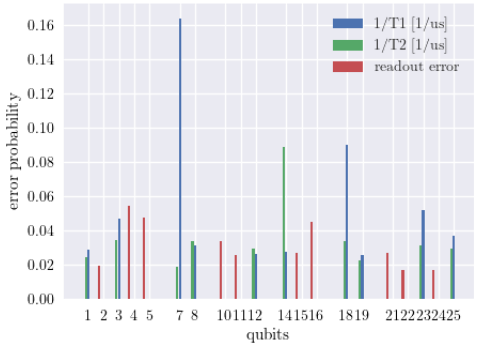
The first step is to analyse weather syndrome-derived single qubit error rates correlate with the properties derived from calibration sequences.



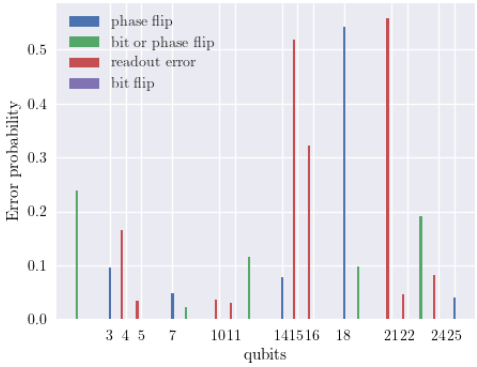
a d=10, T=100, naive algorithm



b d=10, T=100, Spitz algorithm



c Property derived error rates



d d=9, T=100, naive algorithm

Figure 3.18: Microbenchmark analysis for ibmq_mumbai: a, b, d) Syndrome-derived error rates per syndrome cycle for each qubit involved in the ARC circuit. Red bars are readout errors, which can only be measured on ancilla qubits. Blue and green are the error rates of the data qubits. Blue represents phase flips, and green represets data from the Y basis, therefore it can be both, either a bit flips or phase flips. In c) error rates are calculated from the quantum computer qubits T1 and T2 times. The calculation involves to turn the thermal relaxation times into error rates per syndrome cycle, and the readout errors are directly available.

Since a single qubit has just one specific role per circuit execution, only certain types of error rates are available. For the ancilla qubits only the readout error probability is available. On data qubits from the X basis, the phase flip rate is deduced and on Z basis qubits the bit flip rate is calculated. On Y basis qubits only the error rate is available, but it cannot be deduced which kind of error occured. As described in the Theory Section 2.6, two different algorithms (naive approximation, and one developed by Spitz et al. (2018) [30]) were employed to calculate the microbenchmarks. Figure 3.18a shows the

error probability per qubit for ibmq_mumbai from a $d = 10$ and $T = 100$ ARC circuit execution, calculated with the so-called naive algorithm. Figure 3.18b shows the same run calculated with the Spitz algorithm. Figure 3.18c displays a similar graph, but the values were derived from the IBM Quantum backend, thus they represent the theoretical single-qubit values derived from specific characterization and and calibration procedures. The thermal relaxation times are transformed into error rates per QEC cycle with the following Equation 3.1:

$$p_e = \frac{1}{Tn} \times \tau_c \tag{3.1}$$

in which n stands for 1 or 2. T1 is the thermal relaxation rate and T2 is the dephasing rate.

Note, the values for the single qubits are only derived from their thermal relaxation times. Performed gates and gate execution times of the ARC were not considered for this analysis.

As with the introducing break-even point analysis in Figure 3.1, executing a QEC code introduces many errors and noise to the qubits. Comparing the backend derived values against the error rates obtained from the syndrome data shows no significant overlap. It shows that the pure thermal relaxation time is not a good single-qubit error rate predictor when QEC codes are executed. On the other side, the naive algorithm and the Spitz algorithm agree well with each other.

Figure 3.18d shows another run calculated with the naive algorithm, but this time with the $d = 10$ open loop layout. The main difference in the layout is, that the ancilla with the id 2 was removed, with the purpose of not measuring anymore the data qubits with id 1 and 3. Interestingly, removing this ancilla increased the error rate of both neighboring data qubits compared to the closed loop layouts analyzed with the naive and the Spitz algorithm. This is an indicator that the reason, why the closed loop layouts perform worse (as discussed in Section 3.9) is not caused by more increased noise from more qubit-qubit interaction. In this case it is likely a problem on the level of the decoder.

3.12 Using the ARC for diagnostics and benchmarking

Following up on the considerations about QEC benchmarks and diagnostics from Section 1.2, a benchmarking protocol is proposed. This protocol should serve as a basis of discussion to finally qualify the ARC as a candidate diagnostics routine in Quantum Computing Application benchmarking suite like the one from QED-C [9].

3.12.1 Proposal for a benchmarking protocol

The volumetric design of the result reporting of a specific algorithm makes it straightforward for the ARC to conform, since width (the distance d) and depth (number of QEC cycles T) of the code are the two main parameters used throughout this thesis. Usually, the algorithms in the benchmarking suites are executed for different code depths and widths, and a success criterion is evaluated. The success criteria of the ARC is set to

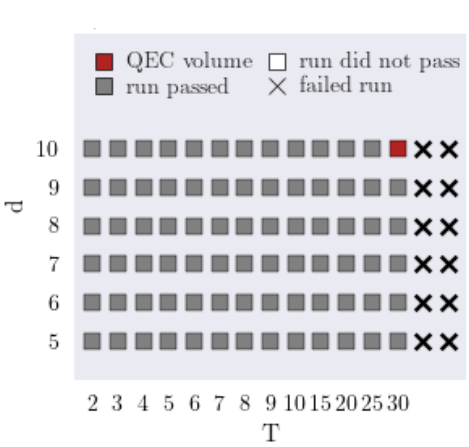
have a logical error rate below $P_e = 0.3$.

In this proposal, the ARC benchmarking is performed with a random linear chain, with varying and randomly selected paths through the defined and device-specific qubit connectivity graph. The maximum distance d_m, and maximum used QEC repetitions T_m can be chosen by the operator and are reflected in the final outcome. The protocol is defined as follows:

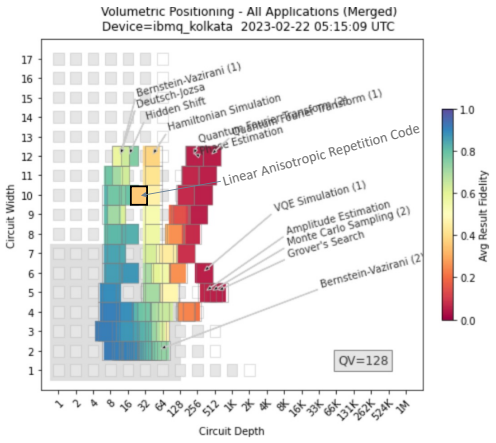
1. The operator provides the maximum distance d and maximum number of repetitions T, the number of runs N, and the possible connectivity graph of the targeted quantum computer
2. Preprocessing: One run is automatically created with the following steps (repeated N times)
   a) Random selection of a qubit in the connectivity graph as starting point
   b) Random walk along the connectivity graph for a random number of steps, up until defined max distance d. (Crossings are not possible, and homogenous distribution should be aimed at by the script)
   c) Random selection of the basis combinations: XY, XZ, YZ, (or use always all possibilities)
   d) Random selection of the logical state (or use always both)
3. Run the prepared runs for $T = 1$ to T_m times
4. Decode the results with maximum of automatic tweaks allowed, to take full advantage of the quantum controller hardware or decoding stack and/or use a predefined standard decoder to assess only the quantum computers' QEC capabilities, (omit any controller-hardware, post-processing, or decoding optimization)
5. Analyse it for $P_e <= 0.3$ and report which combinations of d and T performed successfully.
6. Output the end result consisting of (d, T) pairs with P_e

The analysis part of the protocol was performed on a given dataset, and the output is displayed in the volumetric design, as shown in Figure 3.19a. The best-performing combination of d and T was marked in red, and denoted as QEC Volume. Mind, the data generation of this analysis did not follow the proposed protocol. The full parameter space was mapped and evaluated, and the linear chain path and starting point were also not selected randomly. Admittedly, on the Falcon 27 qubit layout, the number possible different paths is very limited.

Figure 3.19b displays how the ARC fits into the volumetric positioning chart which is the final output of the QED-C quantum application benchmark. The ARC is denoted as Linear Anisotropic Repetition Code and is positioned at a circuit width of 10 and a circuit depth of almost 32 with an orange color indicating a fidelity of 0.33. During an official run with random parameters, many such colored quadratic rectangles are placed in this chart, according to successfully executed runs and their fidelity.



a Volumetric positioning charts of a run of the ARC on ibmq_kolkata. This chart shows, that mainly technical reasons limited executions of more QEC cycles T. Every successfully returned run from the quantum computer passed the success criteria.



b Example of an integration of the ARC could look like in a volumetric positioning comparison chart of the QED-C application benchmark at circuit width of 10 and depth of 31. This chart was thankfully taken and modified from [13].

3.12.2 QEC diagnostics

Apart from the integration into a quantum application benchmark, the proposal regarding QEC-specific diagnostics extends further.

To allow a better comparability between different quantum computers with the ARC as the basis of the diagnostics, the volumetric positioning with its success criteria is used, to limit the window of analysis. This provides a better comparability because many points of large T with bad P_e influence the fits, and therefore downstream the scaling factors ρ and Λ.

An additional step is to combine the volumetric positioning from the QEC Benchmark proposal with the two scaling factors, and put it into one single benchmarking number.

This single benchmarking number B_{QEC} would be defined as:

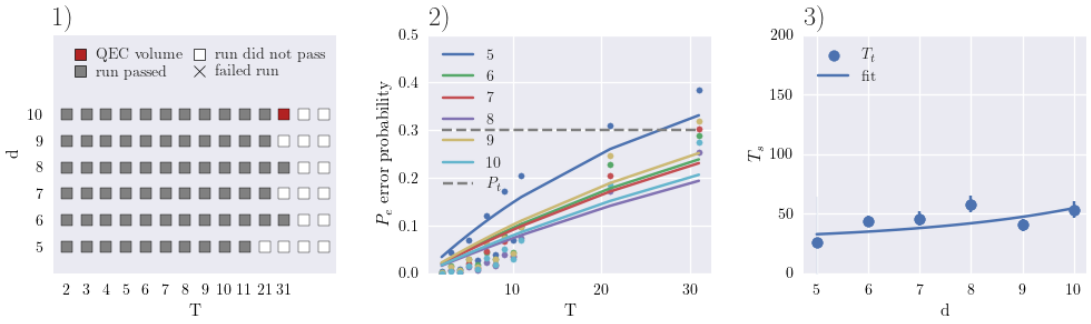
$$B_{QEC} = C_1(1/\rho) - 1 + C_2\Lambda + C_3\sqrt{d_{max}^{2/3} + T_{max}} \tag{3.2}$$

were as d_{max} and T_{max} are the maximum numbers of d and T in which the success criterion was met and $V_{QEC} = \sqrt{d_{max}^{2/3} + T_{max}}$.

C_N are constant factors to balance the impact of the three benchmarking numbers ρ, Λ and V_{QEC} to 1/3 each, and normalize the B_{QEC} to 1 for ibmq_mumbai.

Quantum Computers which perform better in quantum error correction yield a higher number. The range from the current experiments is between 0.7 and 1.58.

This protocol is only a proposal and is exclusively tested with data from QEC Diagnostics runs, so without the randomized preparations. The factors C_N are up to discussion. C_1 and C_2 stand for the scalability of the lifetime and the errors per cycle with the



ρ	0.550
T_o	26.304
A	1.05
Λ	1.201
C	0.022
V_{QEC}	31.0
$(d,T)_{QEC}$	(10,31)
B_{QEC}	1.34

d	T_s	ε_l
5	26.17 ± 3.14	0.017 ± 0.002
6	43.78 ± 4.81	0.01 ± 0.001
7	45.89 ± 5.68	0.01 ± 0.001
8	57.98 ± 7.12	0.008 ± 0.001
9	40.55 ± 4.94	0.011 ± 0.002
10	53.35 ± 6.81	0.009 ± 0.001

Figure 3.20: 1) Volumetric positioning chart to deduce the QEC Volume 2) Apply the QEC Volume to restrict the window of analysis 3) Fit for ρ 4) Fit for Λ

distance. Both factors are important measures to assess, how well a quantum computer shows desired behavior towards quantum error correction. The third factor C_3 accounts for the actual measured error rates within the systematical boundaries like the number of available qubits of a system, or the number of possible QEC rounds which the controller hardware is able to execute.

Figure 3.20 shows an example of such an analysis. The first subplot in a) shows the Volumetric QEC plot, which displays the selected values of d and T, and for which combination of d and T the logical error rate P_e was low enough to pass the success criterion. The QEC Volume is calculated for each run and the highest QEC Volume is selected for bounding the analysis of the logical error rate in subfigure b). Downstream, the scaling factor ρ and Λ are calculated in subfigures c) and d). This two factors are also an input for calculating B_{QEC}. It is not always the case, that the biggest distance, is the most successful. This can be caused by local circumstances like error prone qubits, leakage, heating effects, or overwhelmed controller hardware, as described for the large repetition codes in Section 3.2.

3.13 Leakage analysis

There are two kinds of analyses introduced in Section 2.5 which might detect leakage. The first method is based on syndrome data statistics and the second method uses enhanced readout methods on the quantum computer combined with specific processing procedures to discriminate the states. The experiment to verify the syndrome-based

leakage detection proceeds as follows. For ibm_auckland, level-1 data was acquired and the points on the imaginary plane were discriminated into $|0\rangle$, $|1\rangle$, and $|2\rangle$ states, whereas $|2\rangle$ is an undesired leakage state. The result was stored in three different bit-based syndrome data sets. The first data set contains all three states. For the second data set all shots that contained leakage states were omitted (post-selected dataset). The last contains leakage states that were replaced by 1, to imitate the syndrome data, as they are currently output by the controller hardware (original dataset). The post-selected, and original syndrome datasets were decoded with the usual methods, and a syndrome-based leakage analysis was performed on both. The outcome of the leakage analysis is provided in the following Figure 3.21.

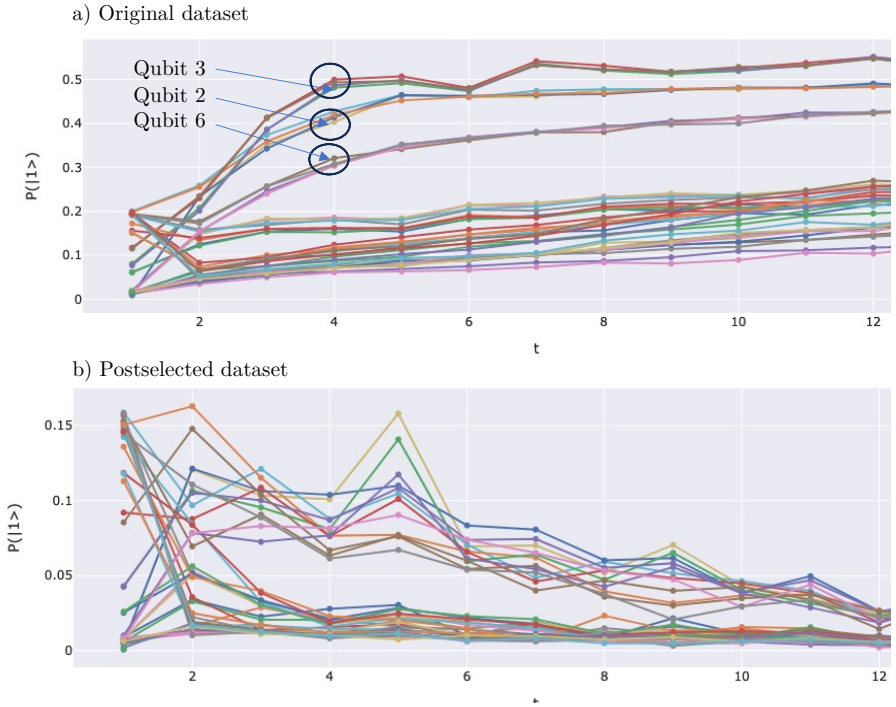


Figure 3.21: a) Leakage analysis of the original dataset, where leakage states are usually counted to the state 1. The Qubits 3, 2, and 6 stand out above the rest. Data traces for Qubit 3 even surpasses the 0.5 threshold. This is a clear indication of leakage. b) The same leakage analysis, but with the post-selected syndrome dataset. The three qubits 3, 2, and 6 do not stand out, and stay in the bulk of the other qubits.

In Figure 3.21 a) the original dataset is analyzed. There are the qubits 3, 2, and 6 which are outliers and approach $P(|1\rangle) = 0.5$. The traces for qubit 3 even overshoot this threshold. Note, that each qubit has four traces because four circuits were executed for varying parameters. The parameters are the logical states $|0\rangle$ and $|1\rangle$ and the basis pairs XZ and ZX. In the Figure 3.21 b) the same analysis is performed on the post-selected dataset and there are no standouts anymore. Therefore, a dataset cleared of leakage, has no traces quickly approaching or surpassing 0.5. The initial hypothesis was, that

only traces that overshoot 0.5 indicate leakage. Post-selected traces approaching this threshold in a short amount of cycles T are completely absent. This leads to loosening the bounds of the hypothesis, such that any qubit which accumulates 1's very fast (and therefore are outliers in this plot), might be leaking.

In summary, there is a strong indicator that this simple syndrome-based analysis method works. More work should be invested to prove this point, to quantify the amount of leakage per qubit and maybe even to develop a method to detect in which round T leakage occurs with the intention to tune the decoding graph [18].

Chapter 4

Conclusions

The Anisotropic Repetition Code (ARC) was introduced with the purpose of benchmarking and diagnosing quantum computers on their ability to perform quantum error correction (QEC).The two factors ρ and Λ were analyzed which quantify the exponential behavior in the lifetime and logical error suppression per round. This metric should help to develop and assess quantum computers towards fault-tolerant quantum computing. There exist multiple quantum computing benchmarking frameworks, but none of them include an easy-to-use and easy-to-interpret application benchmark for quantum error correction which is indicative of more complex QEC codes like the surface code. The ARC is especially useful for this exact purpose because it encodes different bases in neighboring data qubits which emulates more closely the behavior of more complex QEC codes.

The ARC was introduced by establishing where the ARC stands regarding the QEC break-even point on the used quantum computers. At least with a distance of $d = 10$, the ARC is operating past the break-even point. The ARC was introduced and many properties of it were discussed including its behaviour on very large quantum computers, with distances of $d > 50$ on ibm_sherbrooke, and $d = 148$ on ibm_seattle. Several parameters were studied for their influence like the distance d, number of QEC cycles T, the choice of the basis, the choice of the decoder, the usage of resets after the mid-circuit readout or the variability of the scaling factors over time. For all the assessed quantum computers, a larger distance proved beneficial for the scaling, up to a certain point of an optimal distance. Past this point the benefits get lost again. The number of successful repetition cycles T is either limited by the quantum computing controller hardware due to the circuit size, or the logical error probability approaches randomness before the quantum computer finds technical limits in circuit size. Depending on the quantum computing model $T > 150$ were observed or the limits were found at $T = 30$ to $T = 60$. Usually, on quantum computers with limited T the quanum information could be retained over all possible cycles. Using the XY basis posed better results on almost every quantum computer over the XZ basis. XZ seems to be more error sensitive, especially in situations in which leakage occured. With a well tuned decoding graph from syndrome-derived error rates, the PyMatching decoder outperformed the Bravyi-Haah decoder in terms of runtime and decoding quality. Using resets after the mid-circuit measurement limits the number of possible runs T, but poses no significant improvement for the scaling factors, except in the case of strong leakage. In terms of the lifetime measured in microseconds, the quantum information might hold slightly longer,

due to longer circuit execution induced by the resets. The variability of the results over several weeks posed only a significant change in the scaling factors when certain qubits permanently went out of service. A comparison of the scaling factors, the quantum volume, the thermal relaxation times and gate error rates of many quantum computers were conducted and ibm_sherbrooke is best suited to run quantum error correction based on the scaling factors, followed by ibm_hanoi and ibmq_kolkata. The most important factors to predict the ability of a quantum computer to run QEC codes are low readout error rates, and good thermal relaxation- and dephasing times.

Several methods for creating the syndrome data were studied, apart from running the circuits on the quantum computer. The first is called truncation, in which the data is only acquired on the quantum computer for one large distance d, and the rest of the data traces for smaller d are calculated in post-processing, by cutting away the bits, from the syndrome data of the large distance run. This method of truncation did resemble the behavior of the real quantum computer, at least when the one-sided truncation is applied from the same side, as qubits were added to the chain for the real circuit runs on the quantum computer. Tests on truncation indicated, that it generates more homogeneous data compared to the full data set in which every distance was acquired on the quantum computer. The one-sided truncation follows closer the approach of how the actual data is acquired on the quantum computer, by increasing the distance qubit by qubit for every run and not by averaging smaller codes over all qubits. The other method was using simulations with error models derived from the properties of the quantum computers. The simulations produced largely better values of the scaling factors, meaning they do not accurately simulate the real quantum computers performance, with differences between 30% and 60%. Quantum computers face error processes, which are not taken into account by the simulator models like leakage, which pose a significant challenge to the decoder. To detect leakage, a method was developed that analyses the statistics of the ancilla bits in the syndrome data. Tests were conducted, which show that this analysis is suitable to detect leaking qubits.

A protocol was proposed with which the ARC might qualify for being taken up in application-based quantum computing benchmarks for assessing the QEC capability of a quantum computer.

Finally, microbenchmarks were studied, which assess the error rates of single qubits based on the syndrome data. These error rates were successfully used to improve the decoding quality of the Matching decoder.

There are many topics left open for further studies. Leakage seems to be a major topic in QEC. Therefore, it would be beneficial to study the syndrome based leakage detection, try to prove it, develop a quantification of leakage, and explore the possibility to tune the decoding graph based on this information. As an alternative, improving the readout mechanisms of the IBM quantum backend hardware to optionally support the readout of leakage would be of great help to tune the decoding graph to handle the leakage more accurately. Furthermore, it would be beneficial to find a model which supports leakage to fit P_e points in the scaling analysis.

To better understand the ARCs properties, it would be of interest to study influence

of the logical states separately and to investigate the influence of the number of shots.

The ARC was developed to serve as a benchmarking and diagnostics tool. A step foreward in establishing the ARC would be to implement the benchmarking preprocessing protocol and integrate it to the QED-C Benchmarking Suite with a pull request on their Github repository [40].

The investigation of large repetition codes reviled local maxima in the lifetime. This indicates the existance of an optimal distance d_{opt}. It would be interesting to reproduce this phenomena more reliably and investigate the causes of it. If d_{opt} exists, it would be interesting to study the concatenation of multiple codes of optimal distance.

Bibliography

1. Knill, E. *et al.* Randomized Benchmarking of Quantum Gates. *Physical Review A* **77,** 012307. ISSN: 1050-2947, 1094-1622. arXiv: 0707.0963[quant-ph]. http://arxiv.org/abs/0707.0963 (2023) (Jan. 8, 2008).
2. Magesan, E., Gambetta, J. M. & Emerson, J. Characterizing Quantum Gates via Randomized Benchmarking. *Physical Review A* **85,** 042311. ISSN: 1050-2947, 1094-1622. arXiv: 1109.6887[quant-ph]. http://arxiv.org/abs/1109.6887 (2023) (Apr. 11, 2012).
3. Magesan, E., Gambetta, J. M. & Emerson, J. Scalable and Robust Randomized Benchmarking of Quantum Processes. *Physical Review Letters* **106,** 180504. ISSN: 0031-9007, 1079-7114. https://link.aps.org/doi/10.1103/PhysRevLett.106.180504 (2022) (May 6, 2011).
4. Proctor, T., Rudinger, K., Young, K., Nielsen, E. & Blume-Kohout, R. Measuring the capabilities of quantum computers. *Nature Physics* **18,** 75–79. ISSN: 1745-2473, 1745-2481. https://www.nature.com/articles/s41567-021-01409-7 (2022) (Jan. 2022).
5. Cross, A. W., Bishop, L. S., Sheldon, S., Nation, P. D. & Gambetta, J. M. Validating quantum computers using randomized model circuits. *Physical Review A* **100,** 032328. ISSN: 2469-9926, 2469-9934. arXiv: 1811.12926[quant-ph]. http://arxiv.org/abs/1811.12926 (2022) (Sept. 20, 2019).
6. Moses, S. A. *et al. A Race Track Trapped-Ion Quantum Processor* May 16, 2023. arXiv: 2305.03828[quant-ph]. http://arxiv.org/abs/2305.03828 (2023).
7. Gambetta, J. M. *Quantum-centric supercomputing: The next wave of computing* IBM Research Blog. https://research.ibm.com/blog/next-wave-quantum-centric-supercomputing (2023).
8. Aaronson, S. Turn down the quantum volume. https://www.scottaaronson.com/blog/?p=4649 (Feb. 17, 2023).
9. Lubinski, T. *et al. Application-Oriented Performance Benchmarks for Quantum Computing* Jan. 9, 2023. arXiv: 2110.03137[quant-ph]. http://arxiv.org/abs/2110.03137 (2023).
10. Mills, D., Sivarajah, S., Scholten, T. L. & Duncan, R. Application-Motivated, Holistic Benchmarking of a Full Quantum Computing Stack. *Quantum* **5,** 415. ISSN: 2521-327X. arXiv: 2006.01273[quant-ph]. http://arxiv.org/abs/2006.01273 (2023) (Mar. 22, 2021).

11. Tomesh, T. *et al. SupermarQ: A Scalable Quantum Benchmark Suite* in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA) (IEEE, Seoul, Korea, Republic of, Apr. 2022), 587–603. ISBN: 978-1-66542-027-3. https://ieeexplore.ieee.org/document/9773202/ (2022).
12. Vandeth, D., Wootton, J., Harper, G. & Cross, A. *Qiskit QEC* https://github.com/qiskit-community/qiskit-qec (2023).
13. Amico, M. *et al. Defining Standard Strategies for Quantum Benchmarks* Mar. 3, 2023. arXiv: 2303.02108[quant-ph]. http://arxiv.org/abs/2303.02108 (2023).
14. Gottesman, D. *Stabilizer Codes and Quantum Error Correction* May 28, 1997. arXiv: quant-ph/9705052. http://arxiv.org/abs/quant-ph/9705052 (2023).
15. Nielsen, M. A. & Chuang, I. L. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, 2010).
16. Google Quantum AI *et al.* Suppressing quantum errors by scaling a surface code logical qubit. *Nature* **614,** 676–681. ISSN: 0028-0836, 1476-4687. https://www.nature.com/articles/s41586-022-05434-1 (2023) (Feb. 23, 2023).
17. Kelly, J. *et al.* State preservation by repetitive error detection in a superconducting quantum circuit. *Nature* **519.** Number: 7541 Publisher: Nature Publishing Group, 66–69. ISSN: 1476-4687. https://www.nature.com/articles/nature14270 (2023) (Mar. 2015).
18. McEwen, M. *et al.* Removing leakage-induced correlated errors in superconducting quantum error correction. *Nature Communications* **12.** Number: 1 Publisher: Nature Publishing Group, 1761. ISSN: 2041-1723. https://www.nature.com/articles/s41467-021-21982-y (2023) (Mar. 19, 2021).
19. Hsieh, M.-H. & Gall, F. L. NP-hardness of decoding quantum error-correction codes. *Physical Review A* **83.** https://doi.org/10.1103%2Fphysreva.83.052331 (May 2011).
20. Blume-Kohout, R. & Young, K. C. A volumetric framework for quantum computer benchmarks. *Quantum* **4,** 362. ISSN: 2521-327X. arXiv: 1904.05546[quant-ph]. http://arxiv.org/abs/1904.05546 (2023) (Nov. 15, 2020).
21. Higgott, O. *PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching* July 12, 2021. arXiv: 2105.13082[quant-ph]. http://arxiv.org/abs/2105.13082 (2023).
22. Bravyi, S. & Haah, J. Analytic and numerical demonstration of quantum self-correction in the 3D Cubic Code. *Physical Review Letters* **111,** 200501. ISSN: 0031-9007, 1079-7114. arXiv: 1112.3252[cond-mat,physics:quant-ph]. http://arxiv.org/abs/1112.3252 (2023) (Nov. 12, 2013).
23. Kanazawa, N. *et al. T1 Characterization — Qiskit Experiments 0.5 0.5.2 documentation* https://qiskit.org/ecosystem/experiments/manuals/characterization/t1.html (2023).

24. Ezzell, N., Pokharel, B., Tewala, L., Quiroz, G. & Lidar, D. A. *Dynamical decoupling for superconducting qubits: a performance survey* May 1, 2023. arXiv: 2207.03670[quant-ph]. http://arxiv.org/abs/2207.03670 (2023).

25. Ofek, N. *et al.* Extending the lifetime of a quantum bit with error correction in superconducting circuits. *Nature* **536.** Number: 7617 Publisher: Nature Publishing Group, 441–445. ISSN: 1476-4687. https://www.nature.com/articles/nature18949 (2023) (Aug. 2016).

26. Krinner, S. *et al.* Realizing Repeated Quantum Error Correction in a Distance-Three Surface Code. *Nature* **605,** 669–674. ISSN: 0028-0836, 1476-4687. arXiv: 2112.03708[cond-mat,physics:quant-ph]. http://arxiv.org/abs/2112.03708 (2023) (May 26, 2022).

27. Kanazawa, N. *et al. T2* Ramsey Characterization — Qiskit Experiments 0.5 0.5.2 documentation* https://qiskit.org/ecosystem/experiments/manuals/characterization/t2ramsey.html (2023).

28. Kanazawa, N. *et al.* Qiskit Experiments: A Python package to characterize and calibrate quantum computers. *Journal of Open Source Software* **8,** 5329. ISSN: 2475-9066. https://joss.theoj.org/papers/10.21105/joss.05329 (2023) (Apr. 19, 2023).

29. Sundaresan, N. *et al.* Demonstrating multi-round subsystem quantum error correction using matching and maximum likelihood decoders. *Nature Communications* **14.** Number: 1 Publisher: Nature Publishing Group, 2852. ISSN: 2041-1723. https://www.nature.com/articles/s41467-023-38247-5 (2023) (May 18, 2023).

30. Spitz, S. T., Tarasinski, B., Beenakker, C. W. J. & O'Brien, T. E. Adaptive Weight Estimator for Quantum Error Correction in a Time-Dependent Environment. *Advanced Quantum Technologies* **1,** 1800012. ISSN: 2511-9044. https://onlinelibrary.wiley.com/doi/abs/10.1002/qute.201800012 (2023) (2018).

31. Greenbaum, D. & Dutton, Z. Modeling coherent errors in quantum error correction. *Quantum Science and Technology* **3.** Publisher: IOP Publishing, 015007. ISSN: 2058-9565. https://dx.doi.org/10.1088/2058-9565/aa9a06 (2023) (Dec. 2017).

32. Google Quantum AI *et al.* Exponential suppression of bit or phase errors with cyclic error correction. *Nature* **595,** 383–387. ISSN: 0028-0836, 1476-4687. http://www.nature.com/articles/s41586-021-03588-y (2022) (July 15, 2021).

33. Gidney, C., Newman, M., Fowler, A. & Broughton, M. A Fault-Tolerant Honeycomb Memory. *Quantum* **5,** 605. ISSN: 2521-327X. arXiv: 2108.10457[quant-ph]. http://arxiv.org/abs/2108.10457 (2022) (Dec. 20, 2021).

34. Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing* 2023.

35. *IBM Quantum* IBM Quantum. https://quantum-computing.ibm.com/services/resources (2023).

36. Wootton, J. R. & Loss, D. Repetition code of 15 qubits. *Physical Review A* **97,** 052313. ISSN: 2469-9926, 2469-9934. https://link.aps.org/doi/10.1103/PhysRevA.97.052313 (2022) (May 10, 2018).

37. Wootton, J. R. Hexagonal matching codes with two-body measurements. *Journal of Physics A: Mathematical and Theoretical* **55,** 295302. ISSN: 1751-8113, 1751-8121. https://iopscience.iop.org/article/10.1088/1751-8121/ac7a75 (2022) (July 22, 2022).

38. Egger, D. J. & Haupt, C. *Restless simulator* original-date: 2023-01-23T09:16:25Z. Apr. 15, 2023. https://github.com/eggerdj/restless-simulator (2023).

39. Haupt, C. J. & Egger, D. J. *Leakage in restless quantum gate calibration* Apr. 18, 2023. arXiv: 2304.09297[quant-ph]. http://arxiv.org/abs/2304.09297 (2023).

40. Lubinski, T. *et al. Application-Oriented Performance Benchmarks for Quantum Computing* original-date: 2021-09-30T18:00:39Z. Oct. 2021. https://arxiv.org/abs/2110.03137 (2023).

University of Basel

Faculty of Science

Declaration on Scientific Integrity
(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis: Quantum Error Correction Benchmark and Diagnostics

Name Assessor: Prof. Daniel Loss

Name Student: Milan Liepelt

Matriculation No.: 13-631-999

With my signature I declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

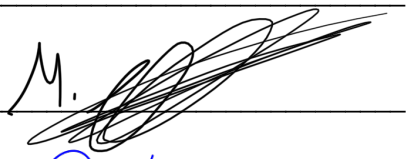
Place, Date: Winterthur, 26. Mai 2023 Student: 

Will this work, or parts of it, be published?

○ No

● Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of: 1. Juni 2023

Place, Date: Winterthur, 26. Mai 2023 Student: 

Place, Date: Basel, 30. Mai 2023 Assessor: D. Loss

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis