Applications of the Quantum Kernel Method on a Superconducting Quantum Processor

by

Evan Peters

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Science
in
Physics (Quantum Information)

Waterloo, Ontario, Canada, 2020

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The widespread benefits of classical machine learning along with promised speedups by quantum algorithms over their best performing classical counterparts have motivated development of quantum machine learning algorithms that combine these two approaches. Quantum Kernel Methods (QKMs) [22, 49] describe one such combination, which seeks to leverage the high dimensional Hilbert space over quantum states to perform classification on encoded classical data. In this work I present an analysis of QKM algorithms used to encode and classify real data using a quantum processor, aided by a suite of custom noise models and hardware optimizations. I introduce and validate techniques for error mitigation and readout error correction designed specifically for this algorithm/hardware combination. Though I do not achieve high accuracy with one type of QKM-based classifier, I provide evidence for possible fundamental limitations to the QKM as well as hardware limitations that are unaccounted for by a reasonable Markovian noise model.

Acknowledgements

My sincere gratitude goes to my MSc adviser Achim Kempf for taking me on as his student as I transitioned into quantum machine learning research. Without his support I would never have been able to explore the field and take part in the collaboration that would eventually culminate in this project. I also thank the members of my committee David Gosset, Robert Mann, and Christine Muschik for their wonderful input that helped shape this master's project into its final form.

Much of this work was completed while I was interning at Femilab National Accelerator Laboratory; this project would not have been possible without my collaborator and frequent host at Fermilab, Gabe Perdue, who has helpfully organized the project and the access to hardware providers necessary to produce the main findings. I'm also indebted to Spence Panagiotis (Fermilab) for his hospitality and my former adviser Heidi Schellman for introducing me to the wonderful Fermilab community.

Other members of a collaboration have contributed to this work throughout, including Joao Caldeira and Brian Nord (Fermilab) who were responsible for preparing, compressing, and studying the datasets used as inputs to the quantum algorithm, Stefan Leichenauer (Alphabet-X) whose higher level approaches contributed to better understanding of the algorithm, and Stavros Efthymiou (Alphabet-X intern) who contributed heavily to the circuit structures, gridsearch methodologies, and some initial hardware runs to pave the way for the results presented herein.

I would also like to thank the members of the Google AI quantum lab who provided us a head start at using the hardware and helped us understand some of the pesky NISQ dynamics taking place on Rainbow-23. Among those who helped were Alan Ho and Masoud Mohseni, who provided insight and set goals as the project progressed, and especially Doug Strain, who was our hardware liason and go-to source for hardware calibration data, updates on the hardware availability, and contributions to our code to make experiments run more smoothly. Of course the hardware and software interface we relied on would not exist without the efforts of the broader Google and Cirq team, most of whom are identified in the author list of [3].

Finally, I am indebted to a broader community of quantum researchers and friends who have provided useful input as I progressed through this project, including Jeremy Bejanin, Galit Anikeeva, Josh Ruebeck, and others, and to my family for supporting me the whole way.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of quantum computing

Quantum computing broadly refers to the use of quantum states (as opposed to classical, binary bits) to perform some computational task, and quantum computers are the hardware purposed to do these tasks. Typically an algorithm on a universal quantum computer (that is, a quantum computer that is at least as powerful as any other quantum computer) is described in terms of operations (or "gates") applied to the underlying quantum state and implemented with some very small amount of error, and the quantum state itself does not decohere due to interactions with the ambient environment. In this case, each step in the evolution of a quantum state $|\psi\rangle$ can be described exactly by applying a unitary operator U to a vector representation of the state.

In the gate model for quantum computing [36] a quantum algorithm is represented as a circuit diagram with the component qubits of $|\psi\rangle$ represented as lines moving forwards (to the right) through time and being acted on by unitaries represented by closed boxes (see Figure 1.1). This representation is useful for connecting the mathematical description of unitary operators acting on complex Hilbert space with the physical evolution of a state $|\psi\rangle$ in time and the classical notion of bits being acted on by local logic gates.

There are many proposed physical implementations for near-term quantum computers, and all of them are susceptible to physical defects, environmental noise, and control errors that interfere with the device's ability to perfectly execute the operations depicted in a quantum circuit. When quantum algorithms are studied theoretically, the specific nature of these errors arising from any given device is typically abstracted away, and only the

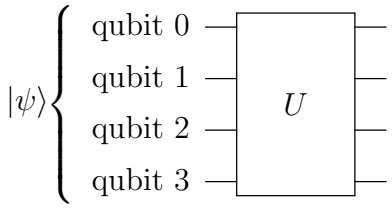


Figure 1.1: A generic quantum circuit for evolving an initial state $|\psi\rangle \in \mathbb{C}^{16}$ via operator U.

intended evolution of the system's wave function (such as the representation in Figure 1.1) is considered. The goal of quantum error correction is to achieve this idealization by introducing additional operations that prevent the effects of noise that would otherwise ruin the computation; a quantum system that can carry out computation even in the presence of imperfect gates and noise is said to be *fault tolerant* [40].

To date, the most promising quantum algorithms are those designed to run on fault tolerant, universal quantum computers and which solve a problem in fewer steps than the best-performing, currently known classical method. A quintessential example is Shor's algorithm for factoring integers [51], which can find prime factors of an integer N in an amount of time that scales like $O(\log(N)^3)$ (the big "O" describes asymptotic time cost with respect to a very large N); meanwhile, to date the best classical algorithm takes almost exponentially longer time (with respect to N) to perform the same task. Another famous example is Grover's search algorithm [19], which, given a suitable quantum implementation of a binary function $f : \{0, 1\}^n$, determines if the function outputs "1" with a worst case performance of $O(2^{n/2})$ queries. This is a quadratic speedup over the worst case classical performance of $O(2^n)$, which arises as a result of guessing the input that yields "1" last. Beyond solving classical problems, quantum computers are expected to be able to simulate the dynamics of quantum systems much more efficiently than computing the exact evolution of such states on a classical computer [17].

Though certain quantum algorithms achieve speedups over their classical counterparts, a fault tolerant device capable of executing such algorithms doesn't exist yet. Instead, practical quantum computation is currently only available in a very limited form.

NISQ devices

Implementing unitaries with very low error (and therefore achieving state evolution that can be represented in a form like Figure 1.1) typically requires error correction that is too resource intensive to accomplish on current "Noisy Intermediate Scale Quantum" (NISQ)

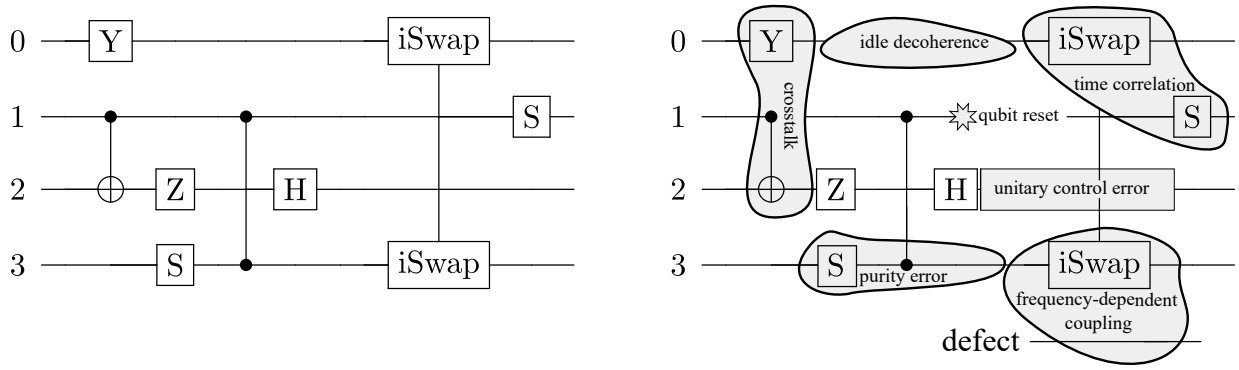


Figure 1.2: (right) A more realistic model for what actually occurs when the quantum circuit on the (left) runs on a NISQ device. Round-edged shapes denote non-unitary effects that are generally neither local in time nor space.

hardware [42] – instead, algorithms implemented on NISQ hardware are limited to tens to hundreds of qubits which steadily decohere due to interactions with the environment and are subjected to unpredictably imperfect evolution.

A more realistic view of dynamics on a NISQ device is presented in Figure 1.2, which depicts many potential non-unitary effects that might influence a quantum state's evolution. Some of these effects can be approximated using a gate model for state evolution, and many can be probed by comparing the results of hardware experiments to the perfect evolution predicted by a corresponding noiseless gate model circuit. Chapters 2-3 are devoted to understanding these underlying dynamics that drive the evolution of a state on a NISQ device.

Unlike algorithms that will eventually be executed on fault tolerant quantum computers, it is unclear whether algorithms executed on NISQ devices will be able to reliably produce speedups over useful classical algorithms. One recent demonstration of this kind of "quantum supremacy" reported successful sampling from a distribution that is inefficient to sample from classically, at a rate much faster than very large classical computers could [3]. At the same time, the distribution that was sampled was only marginally similar to the intended distribution (i.e. the distribution achievable with error correction), which highlights one difficulty of assessing the performance of algorithms on NISQ devices and foreshadows similar difficulties in assessing the algorithm that is the topic of this work (see Section 4.3.3: "On quantum advantage").

Despite the difficulties of demonstrating clear-cut quantum speedup, the increasing availability of NISQ hardware motivates a broader search for algorithms capable of running

on near-term devices. The next section will discuss a surprising contender: a class of algorithms that utilize quantum computers to process classical data.

1.2 Motivations for quantum machine learning

Machine learning (ML) can be broadly described as the use of algorithms that can classify, predict, or interact with data in a generalizable way using large amounts of input data and optimization. ML has gained popularity for its impressive performance in a wide range of fields, from facial/image recognition to high energy physics to natural language processing. This performance comes at a cost, however, as training ML models can consume thousands or even millions of desktop computers worth of processing power.

Given the successes of classical machine learning, a natural extension is to try to combine its advantages with quantum computing. Recently, "Quantum Machine Learning" (QML) has become an umbrella term describing the use of a quantum computer to enhance the performance of a classical ML model or the application of classical ML to improve the performance of quantum circuits. Some examples include the Variational Quantum Eigensolver [30] which uses classical optimization to prepare the ground state of Hamiltonian, the Quantum Approximate Optimization Algorithm [15] which uses classical optimization of parameters in a quantum circuit to solve graph problems, and quantum circuit architectures loosely modelled after popular classical machine learning algorithms like the Quantum Neural Network [16] which uses a quantum circuit to classify input classical data via manipulations in Hilbert space and Quantum Convolutional Neural Network [13] which classifies input quantum states according to their entanglement properties.

The Quantum Kernel Method (QKM) [49, 22], the topic of this thesis, attempts to find clean separations in classical data encoded into a quantum state by taking advantage of the exponentially large size of Hilbert space that governs states over even a few qubits. As shown in Figure 1.3, data that is difficult to linearly separate in low dimensions can become trivial to separate using a ML model such as the Support Vector Machine (SVM) when a specific projection is used to map the data points into higher dimensional space.

Each of these QML algorithms attempts to solve a problem that is difficult to do with classical resources alone, but they all share some degree of difficulty in proving that they offer any scaling or resource advantage over the use of classical algorithms alone. The difficulty of demonstrating such a result for a QML algorithm is compounded by the fact that the field of classical ML algorithms often fails to provide any rigorous proof of performance for its most popular algorithms, instead resorting to comparisons to accepted

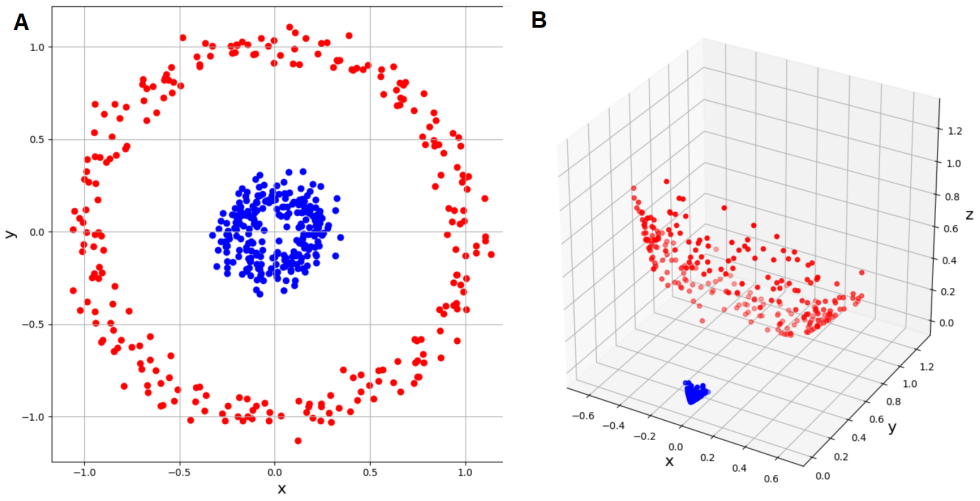


Figure 1.3: The task of finding a simple, generalizable function that separates blue dots from red dots is nontrivial and highly susceptible to overfitting. Conceptually, the power of kernel methods comes from their ability to project these kind of data that are not linearly separable in their original form (image A) into higher dimensional space (for example, image B) which allows for a simple and generally accurate dividing hyperplane to be drawn.

benchmarks or empirical scaling trends to evaluate a model. Chapter 4 is devoted to addressing simulated performance and discusses the issue of "quantum advantage" for QML algorithms and the QKM/SVM combination specifically.

Just as classical neural networks (universal function approximators capable of learning arbitrary classification boundaries in data) have largely eclipsed SVMs due to the former's greater expressiveness over input data (as Chapter 4 will describe, SVMs are limited to classifying data based strictly on inner products between mapped data points), more complex QML algorithms may turn out to be more powerful than the SVM/QKM combination described in this work. To that end, in addition to the work presented here I helped develop a quantum machine learning library called Tensorflow Quantum [11] designed to assist with prototyping and implementation of quantum machine learning algorithms on quantum hardware. With this software and its connections to a massive ecosystem of tools for classical machine learning and optimization, the historically rapid development of ML may yet repeat itself in the quantum domain.

1.3 Overview of this work

This thesis is structured to provide the background and results for noise models for hardware, quantum processor performance using the Rainbow-23 Google superconducting qubit chip, and finally performance of an ML model trained using results from Rainbow-23. However, for the purpose of this work these separate of a single QML model will be addressed and validated separately. Therefore the goals of this project fall into three distinct categories:

1. (Chapter 2) Develop a noise model that is predictive of hardware outcomes.
2. (Chapter 3) Verify and optimize the outputs of quantum hardware.
3. (Chapter 4-5) Maximize the accuracy of an SVM model that employs QKM.

In the spirit of rigorous verification, the outcomes associated with each goal will be addressed separately. To understand why performance in one category doesn't necessarily transfer to another, consider the following examples:

- A circuit-based noise model might accurately predict outcomes from quantum circuits run on quantum hardware even if the degree of decoherence means that the hardware outcomes are sampled from a provably classically tractable distribution, and a detailed understanding of noise processes on hardware won't necessarily assist in developing a robust ML model.
- Results from quantum hardware might be sampled from a coherent state vector in Hilbert space despite a noise model's failure to reconstruct the distribution or despite bad performance of ML models employing this distribution.
- A classical ML model might perform well using results sampled from decohered states on quantum hardware that were essentially acting as a biased random number generator (See Section 4.3.2 for example) or without any prior knowledge of the target distribution provided by a detailed noise model.

Because each of these goals is distinct, the performance of a predictive noise model isn't necessarily correlated with quantum behavior of the Rainbow-23 processor, and the performance of a quantum-assisted ML model. As such, successes (or failures!) in any given category should be viewed in a standalone manner.

Chapter 2

Markovian noise and gate errors in quantum hardware

The goal of this project is to train a machine learning model using the quantum kernel method on the Rainbow-23 superconducting qubit hardware. Before presenting results related to hardware, its important to develop and understanding of noise in superconducting qubit systems and establish a simulated noise model that can closely predict the outputs any given circuit executed on real hardware. Understanding the noise characteristics of a quantum system is essential to this work for the following reasons:

- Making robust predictions about the performance of quantum kernel circuits in the design process.
- Validating optimizations and postprocessing steps meant to undo error introduced by real hardware.
- Determining how well the hardware performance conforms to expectations, and identifying sources of hardware errors that are unaccounted for in a gate-model representation of quantum computing (see Figure 1.2).

This chapter focuses on Markovian noise, which roughly describes the class of physical processes for which a single time step of evolution depends only on the current state of the system[1]. This class of noise is both easier to describe (Section 2.1) and more convenient

[1]In a rigorous treatment the term "Markovian" is typically reserved for describing processes that can be modeled using stochastic matrices (as opposed to unitary processes that use unitary matrices, for example). Here the term will be used more loosely.

to simulate in a quantum circuit simulator setting (Section 2.3) than the alternative class of non-Markovian noise, and the phenomenological nature of the noisy simulation allows for implementation and tuning based on empirical results from the hardware.

2.1 Noise processes

The following sections introduce some conventions for modeling noise processes and describing the performance of imperfect quantum gates, as well as example sources of noise and diagnostics for superconducting quantum hardware. Finally, Section 2.3 presents a full noise model that can be implemented in a gate based quantum circuit simulator.

2.1.1 Noise Channel representations

Noise channels can be represented as completely positive (positive over all possible extensions of the relevant Hilbert space) trace preserving, or CPTP, maps on the space of density matrices. The CPTP restriction enforces that the result of applying a noise channel to a density matrix is itself a valid density matrix. As such, the action of a noise channel is denoted $\mathcal{D} : L(\mathcal{H}) \to L(\mathcal{H})$ since the channel is a mapping between linear operators in a given Hilbert space. This mapping admits many possible representations, two of which will be introduced in the following sections.

Kraus (operator sum) representation

The first is the Kraus operator sum representation, in which the action of a noise channel $\mathcal{D}[p]$ (square brackets denote parametrization by p) is characterized by a set of operators $\{M_i \in L(\mathcal{H})\}$ acting on a density matrix $\rho \in L(\mathcal{H})$ in the following way:

$$\mathcal{D}[p](\rho) = \sum_i M_i(p)\rho M_i^\dagger(p) \tag{2.1}$$

This picture of noisy evolution is convenient because it describes evolution of a quantum state as a mixture unitary evolutions, weighted by probabilities contained in the M_i operators. However, just as density matrix representations of a state are not uniquely defined by summations of pure states, the Kraus representation does not provide a *unique* way of representing the action of a channel $\mathcal{D}$.

Choi matrix representation

The Choi matrix is a unique way of capturing the dynamics of a process, $\mathcal{D}$. To do so, the action of the process is extended to a larger Hilbert space and computed on a maximally entangled state. To prevent ambiguity that arises from this extension, enforce that $\mathcal{D} \in \mathbb{C}^d$. Then the Choi matrix $J(\mathcal{D})$ is defined by the action of the channel in extended Hilbert space on a maximally entangled state [57]:

$$J(\mathcal{D}) = (\mathcal{D} \otimes I)|\Omega\rangle\langle\Omega| \tag{2.2}$$

where $|\Omega\rangle\langle\Omega|$ is the maximally entangled state over $\mathbb{C}^d \otimes \mathbb{C}^d$:

$$|\Omega\rangle \equiv \sum_i^d |ii\rangle \tag{2.3}$$

For example, if $\mathcal{D}$ acts on two-qubit density matrices $\rho \in \mathbb{C}^{4\times4}$ then $d = 4$, and i to iterates on $0, 1, 2, 3$. If i instead iterates over the binary representation $i = 00, 01, 10, 11$, then $|\Omega\rangle \in \mathbb{C}^4$ evaluates to

$$|\Omega\rangle = |0000\rangle + |0101\rangle + |1010\rangle + |1111\rangle$$

The Choi matrix can be computed from the more familiar Kraus-sum form over the Kraus operator set $\{A_m\}_{m=1}^M$. That is, if $\mathcal{D}(\rho) = \sum_m^M A_m \rho A_m^\dagger$, then the Choi representation is equivalently [54]:

$$J(\mathcal{D}) = \sum_m^M (A_m \otimes I)|\Omega\rangle\langle\Omega|(A_m^\dagger \otimes I) \tag{2.4}$$

2.1.2 Measurement error

Section 4.3 will introduce quantum circuits capable of computing quantities from sampled outcome distributions via projective measurement. Experiments in this project were submitted and run on Google hardware remotely and then the sampled distributions were returned, but the presence of errors in the measurement process means that these distributions will contain incorrect entries according to the degree and type of errors. Later sections will develop methods for correcting measurement error described here, and validate these methods by applying them to noisy simulations containing the same types of measurement errors as the hardware experiments.

Measurement in the Rainbow-23 superconducting qubit system involves applying a pulse to a readout resonator at its resonant frequency and determining the qubit state via transmittance [5]. In practice, this is prone to the following errors (and more) [24]:

1. Separation fidelity: Measurement requires a calibration in which the microwave signal phases and amplitudes corresponding to the presence of $|0\rangle$ or $|1\rangle$ are determined. Imperfect identification of points in phase/amplitude space due to finite statistics predisposes the measurement to false negatives and false positives.

2. Relaxation: Measurement occurs over a finite timespan, during which qubits are prone to decay due to the ambient noise conditions in the system (namely T_1, T_2 decay). This has the tendency of driving a qubit state towards the maximally mixed state.

3. Transitions toward equilibrium: At finite temperature T, the number of lower energy states N_0 of a system (i.e. $|0\rangle$) will trend towards an equilibrium distribution described by the Boltzmann ratio

$$\frac{N_0}{N_1} = \exp(\hbar\omega_{01}/kT) \tag{2.5}$$

   where ω_{01} is the transition frequency between states $|0\rangle$ and $|1\rangle$. As a result, the probability of a state $|0\rangle$ being measured as such is suppressed by the possibility of thermal excitation towards equilibrium during the measurement period.

In practice, relaxation is the dominant error affecting readout [24] so that the probability p_1 to observe a '0' readout given the state $|1\rangle$ tends to be larger than the probability p_0 to observe a '1' readout given the state $|0\rangle$. Furthermore, the relaxation effect and the equilibration effect are both decohering processes, and do not cancel eachother out in any way. Rather, the "visibility" $1 - (p_0 + p_1)$ of the system is used to describes the overall readout accuracy.

2.1.3 T1/T2 decay

T_1 and T_2 times refer to the longitudinal and transverse relaxation times for a qubit state [6]. In a Bloch sphere picture, T_1 describes relaxation of the Bloch vector towards the *positive* Z axis, while T_2 describes relaxation towards the Z axis (shrinking the XY-component of a Bloch vector).

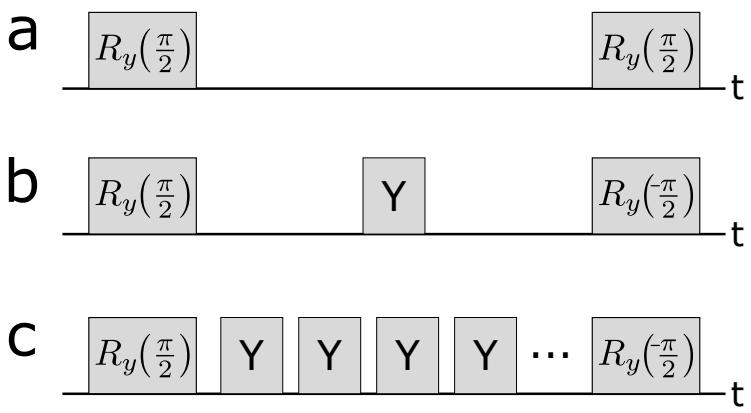


Figure 2.1: Gate sequences for (a) Ramsey T_2 experiment (b) Hahn echo T_2 experiment and (c) CPMG T_2 experiment with $\Delta t = 25$ ns (the time required to implement a Y gate on the Rainbow-23 processor).

The T_2 of a system is typically probed by performing a "Ramsey experiment", wherein the state $|0\rangle$ is rotated into $|+\rangle$, allowed to idle for some amount of time t, and then rotated into $|1\rangle$ and measured [46]. By repeating this process over many different values of t, there will be exponential decay (with slope given by T_2) in the probability $p(1)$ of measuring $|1\rangle$ with respect to time.

If low frequency noise ($f < 1/t$) is present in the system, then its contribution to the decoherence of $|+\rangle$ can be reversed by applying a single Y pulse (sometimes referred to a "Hahn echo" [21]) after a duration of $t/2$. The decay in $p(1)$ will then correspond strictly to noise effects originating from physical processes of frequency ($f > 1/t$), and the state is said to have been "refocused".

The logical extension of a Hahn echo experiment is the CPMG [12, 31] pulse sequence, wherein the evolution of the $|+\rangle$ state is interrupted by n refocusing pulses spaced evenly at time intervals of $\Delta t = t/n$, and therefore including noise effects originating from physical processes of frequency ($f > n/t$) in the calculation of T_2.

Figure 2.1 provides a summary of the gate sequences used to implement each of these T_2 diagnostics in the context of a gate-based quantum computer.

2.2 Gate fidelity diagnostics

2.2.1 Process tomography

Given an extensive set of input states and measurement bases, standard QPT can be used to experimentally determine the form of a noise channel $\mathcal{D}$. Section 2.1.1 showed an explicit conversion from Kraus operators to Choi matrix for a channel, and converting between forms is possible in general [54]. So once a full characterization of a noise process has been determined by some method, the Choi representation or one of its possible Kraus operator representations can be recovered without loss of information.

A standard method for determining the form of a noisy quantum process is Quantum Process Tomography (QPT). In standard applications, QPT is used to determine the fidelity of an experimental quantum gate by treating the application of the gate as a noisy channel, and then computing its difference ("fidelity") compared to the noiseless version of the quantum gate.

Let $\mathcal{D}_U$ be the channel representation of a unitary U plus some coherent and decoherent noise. The fidelity of this gate-as-channel is the quantity as [35]:

$$F(\mathcal{D}_U, U) = \int d(|\psi\rangle\langle\psi|) \operatorname{Tr}\left[U^\dagger \mathcal{D}_{\mathcal{U}}(|\psi\rangle\langle\psi|) U |\psi\rangle\langle\psi|\right] \tag{2.6}$$

where the integral is with respect to the Haar measure $d(|\psi\rangle\langle\psi|)$, which just enables integrals over dense subsets (i.e. sets of states ρ) in Hilbert space. Note that $F(\mathcal{D}_U, U) = 1$ is satisfied if $\mathcal{D}_U(\rho) = U\rho U^\dagger$. On the other hand, if $\mathcal{D}_U$ represents a maximally decohering channel ($\mathcal{D}_U(\rho) = \frac{1}{d}\mathbb{I}$) then the fidelity is $\frac{1}{d}$.

[23] provided a simplified form of Equation 2.6 for the fidelity of a channel provided in Kraus form. This result was then recast for Choi matrix representations in [25], resulting in the formula:

$$F_{U^\dagger \circ \mathcal{D}_U} = \frac{1 + d\,\langle\Omega|\,(I \otimes U^\dagger) J(\mathcal{D}_{\mathcal{U}})(I \otimes U)\,|\Omega\rangle}{d(d+1)} \tag{2.7}$$

where $(U^\dagger \circ \mathcal{D}_U)(\rho) = U^\dagger \mathcal{D}_U(\rho) U$.

In contrast to the methods outlined in the following sections, process tomography provides a complete description of the noise channel that was enacted on a gate during its execution, in its Choi representation or any other representation via a conversion like Equation 2.4. I will take advantage of the thoroughness of this diagnostic in implementing optimization procedures (see Section 3.2.3).

2.2.2 Cross entropy benchmarking

Cross Entropy Benchmarking (XEB) [7] determines gate error by comparing a circuit output's convergence to an expected distribution with its convergence to a random distribution (the metric used for this comparison is defined as "cross entropy"). In the context of the Rainbow-23 hardware, the diagnostic yields a purity error and a total error for two-qubit gates applied to each pair of qubits on the grid.

In general, control error can be expected to introduce a small amount of unitary error during the application of a two-qubit gate (the most basic explanation for this phenomenon is that two qubits with different frequencies will acquire a relative phase proportional to their detuning in addition to any Hamiltonian that is implemented over their combined system). The outcomes of an XEB calibration do not yield specific information about the nature of coherent errors over two-qubit gates. Instead, it is inferred from the difference between a total gate error and the purity error.

2.2.3 Randomized benchmarking

The single qubit gate fidelities on the Rainbow-23 device used are characterized by one-qubit Clifford randomized benchmarking (RB) [27]. RB fidelity is computed by determining the average marginal error introduced into a sequence of Pauli gates due to the addition of single gate to the sequence, and is therefore not specific to the identity of the single qubit gate being implemented and robust to state preparation and measurement errors typically associated with gate fidelity diagnostics. In this work the RB fidelity incorporates both unitary error and decoherence [2].

In [3] some evidence was provided for agreement between RB and XEB fidelities, though to date no rigorous theoretical treatment of this comparison has been done.

2.3 Simulation noise model

Section 2.1 provided theoretical tools to understand decoherence in open systems. While states in open quantum systems undergo evolution, the gate model for quantum computing (and therefore the set of simulatable quantum circuits) is restricted to discrete-time channels and unitary operations. Therefore the goal of this section is to present the components

[2] At the time of writing, diagnostics were being developed for Rainbow-23 that would separate these effects into two different metrics

of a noise model that can be run as a (discrete-time) gate model quantum circuit but which captures much of the noise phenomena that are expected to be present in actual quantum hardware.

As the title of this chapter suggests, this model will be purely Markovian, in that at each "moment" the quantum state evolves according to time-independent noise channels; the only time dependence of the density matrix in the noisy simulation arises from the arrangement of consecutive noise channels.

2.3.1 T1/T2 decay

T1 and T2 decoherence can be simulated using amplitude damping and dephasing channels respectively. From [41], an amplitude damping channel $\mathcal{D}_{AD}$ incurring a de-excitation with probability $\Gamma\Delta t = \Delta t/T_1$ in a time interval $\Delta t << T_1$ leads to an exponential decay in the probability to observe an excited state $\rho_{11}(t)$ like:

$$\left(1 - \frac{\Delta t}{T_1}\right)^{t/\Delta t} \to e^{-t/T_1} \tag{2.8}$$

For gate durations that are orders of magnitude lower than T_1 values, it follows that discrete amplitude damping channels $\mathcal{D}_{AD}$ can be applied to simulate the incoherent idle decay of qubits. The Kraus representation of $\mathcal{D}_{AD}[p]$ parametrized by a decay probability p is given by the following set of operators:

$$M_0 = \begin{pmatrix} 0 & \sqrt{p} \\ 0 & 0 \end{pmatrix}, \qquad M_1 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{pmatrix} \tag{2.9}$$

The T_2 decay of the qubit states due to phase noise can be similarly simulated using a phase damping channel $\mathcal{D}_{PD}[p]$ with the Kraus representation given by the operator set:

$$M_0 = (1-p)^{\frac{1}{2}}\,\mathbf{I}, \qquad M_1 = \begin{pmatrix} \sqrt{p} & 0 \\ 0 & 0 \end{pmatrix}, \qquad M_2 = \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{p} \end{pmatrix} \tag{2.10}$$

Similarly to T_1 decay, this leads to exponential decay in the *off-diagonal* terms of the single-qubit density matrix, ρ_{10} and ρ_{01} [41].

Since the decoherence captured by a fidelity metric on single qubit gates includes the same noise profile as for idle qubits, its important to enforce that $\mathcal{D}_{AD}$ and $\mathcal{D}_{PD}$ are applied only to qubits that are not acted on by a gate (which will be modified by a noise channel separately to capture the gate infidelity). Additionally, these channels are paremtrized according to the *longest* gate time in that moment.

2.3.2 Measurement error

Measurement error can be simulated by classical postprocessing methods such as Monte Carlo simulation of bitflips. However for completeness (namely the ability to generate exact amplitudes for bitstrings in noisy simulations) I implement the measurement error as a channel with the Kraus representation

$$M_0 = \begin{pmatrix} 0 & 0 \\ \sqrt{p_0} & 0 \end{pmatrix}, \qquad M_1 = \begin{pmatrix} 0 & \sqrt{p_1} \\ 0 & 0 \end{pmatrix}, \qquad M_2 = (1-p_0)^{\frac{1}{2}}\,\mathbf{I} \tag{2.11}$$

This has the desired effect of reproducing the bitflips that could otherwise be achieved by classical post-processing while still acting on amplitudes in the full (potentially entangled) quantum state[3].

2.3.3 Single qubit gate error

The state infidelity due to a total RB error of p_e can be simulated by the symmetric depolarizing channel $\mathcal{D}_D[\frac{3}{4}p_e]$ where $\mathcal{D}_D[p]$ has the Kraus representation

$$M_0 = (1-p)^{\frac{1}{2}}\,\mathbf{I}, \qquad M_i = \left(\frac{p}{3}\right)^{\frac{1}{2}}\sigma_i \tag{2.12}$$

where σ_i are the single qubit pauli matrices.

2.3.4 Two qubit gate error

In general the two-qubit error will be a combination of two mechanisms: Purity error and unitary error. Purity error is caused by incoherent noise (which acts to send pure states to mixtures), while the remainder of the error is unitary (and therefore potentially reversible). Unfortunately, even accounting for the non-uniqueness of operator-sum representations, the space of two-qubit decohering channels is very large. In simulations, I used a two-qubit generalization of the symmetric depolarizing channel $\mathcal{D}_{2D}[p]$ to enact the purity error, with a Kraus representation like:

[3] It can be shown that this channel is equivalent to a composition of amplitude damping and bitflip for a certain parametrization

$$M_{ij} = \left(\frac{p}{15}\right)^{\frac{1}{2}} \sigma_i \otimes \sigma_j \qquad ij \neq 00 \tag{2.13}$$

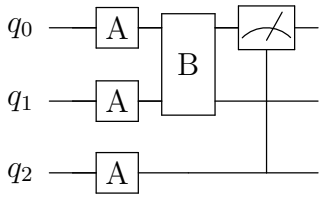
$$M_{00} = \left(1 - \frac{15}{16}p\right)^{\frac{1}{2}} \mathbf{I} \tag{2.14}$$

while to simulate two-qubit gate coherent error of magnitude p, I applied a unitary randomly sampled from the set of operators of the form

$$U(p) = \exp(ip\sigma_i \otimes \sigma_j) \tag{2.15}$$

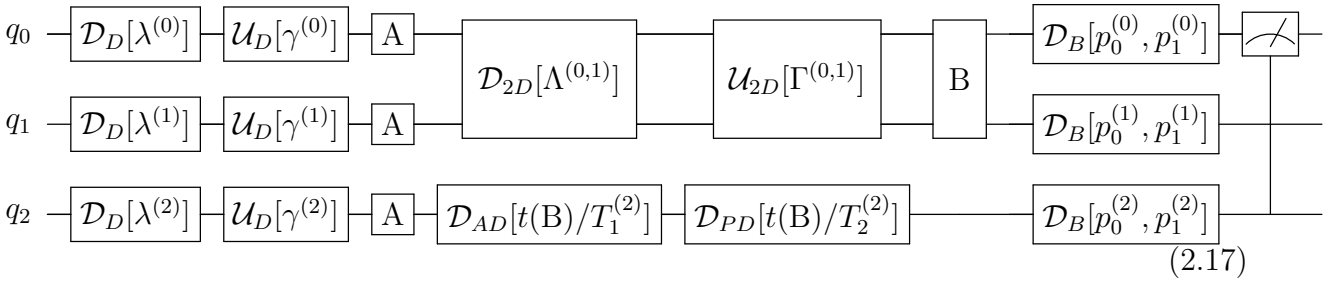
2.3.5 Complete simulated noise model

The following circuits provide a minimal complete example for how the components of the introduced noise model fit into an arbitrary simulated circuit. Beginning with a circuit like the following (A and B represent arbitrary single qubit and two-qubit gates respectively):



(2.16)

then applying the set of time-independent noise channels introduced in the preceding sections would result in the following circuit:



(2.17)

Where the symbols appearing in the circuit model are defined in Table 2.1.

Table 2.1: The noise model consumes a combination of calibration data provided with the Rainbow-23 results, and the results of custom diagnostics.

Symbol	Description
$T_1^{(i)}$	T_1 for qubit i
$T_2^{(i)}$	T_2 for qubit i
$t(X)$	execution time for gate X
$p_0^{(i)}$	Probability of measuring "1" given a state $\|0\rangle$, for qubit i
$p_1^{(i)}$	Probability of measuring "0" given a state $\|1\rangle$, for qubit i
$\lambda^{(i)}$	Single qubit RB purity error for qubit i
$\gamma^{(i)}$	Single qubit RB unitary error for qubit j (Not implemented in this work)
$\Lambda^{(i,j)}$	Two qubit XEB purity error for the qubit pair (i, j)
$\Gamma^{(i,j)}$	Two qubit XEB unitary error for the qubit pair (i, j)

Chapter 3

Hardware diagnostics and optimization on a superconducting quantum processor

This chapter details results of running diagnostics and preparing optimizations for the actual QKM experiments of Chapter 5. Section 3.2.1 describes hardware calibrations that were performed to optimize the performance of quantum circuits with respect to the Rainbow-23 hardware.

3.1 Basic diagnostic data

Periodic calibrations of the Rainbow-23 superconducting qubit device produce diagnostic data describing qubit and gate performances. The following list contains the calibration data that are relevant to this project that the hardware providers release following the submission of each experiment[1]:

- Readout p_0 error: The probability of a computational basis measurement on any given qubit reporting a "1" when the actual result should have been "0".

[1]Use of the Rainbow-23 superconducting qubit device and access to the associated calibration data was facilitated by A. Ho, M. Mohseni, and D. Strain and made possible through the UWaterloo-Fermilab collaboration's partnership with the Google AI Quantum Laboratory.

- Readout p_1 error: The probability of a measurement reporting a "0" when the actual result should have been "1".
- Single qubit T_1: Longitudinal decoherence time for each qubit (see Section 2.1.3)
- Single qubit gate RB error: Average gate error per gate, computed using Randomized Benchmarking (Section 2.2.3)
- $\sqrt{\text{iSWAP}}$ gate XEB total error: The average infidelity for each $\sqrt{\text{iSWAP}}$ two qubit gate according to cross entropy benchmarking (Secction 2.2.2)
- $\sqrt{\text{iSWAP}}$ gate XEB purity error: The decoherence introduced by each $\sqrt{\text{iSWAP}}$ two qubit gate according to cross entropy benchmarking, as opposed to unitary error. The sum of purity error and unitary error is the total error.

These calibration data are used for running the gate-based noise simulation of Section 2.3.5 to predict the performance of submitted circuits and for optimizing qubit selection of Section 5.2.3.

Since T_2 values are necessary for a complete noisy simulation but were not provided with the periodic hardware calibration data, the following section details experiments that I conducted on the hardware to determine values of T_2.

3.1.1 T_2 diagnostics

This section details the results of different diagnostics for T_2. For each qubit, I ran a standard Ramsey experiment, a Hahn-echo experiment, and a CPMG sequence (with the minimum possible wait time of 25 ns - the duration of a Y gate) over all qubits in parallel (see Section 2.1.3 and Figure 2.1 for experiment descriptions). T_2 values were determined using a least-squares fit to exponential decay, slightly modified from [38]:

$$P_1(t) = A \exp\left[\left(\frac{t}{T_2}\right)^B\right] + C \tag{3.1}$$

Figure 3.2 shows the results of performing fitting like Figure 3.1 for each qubit on Rainbow-23.

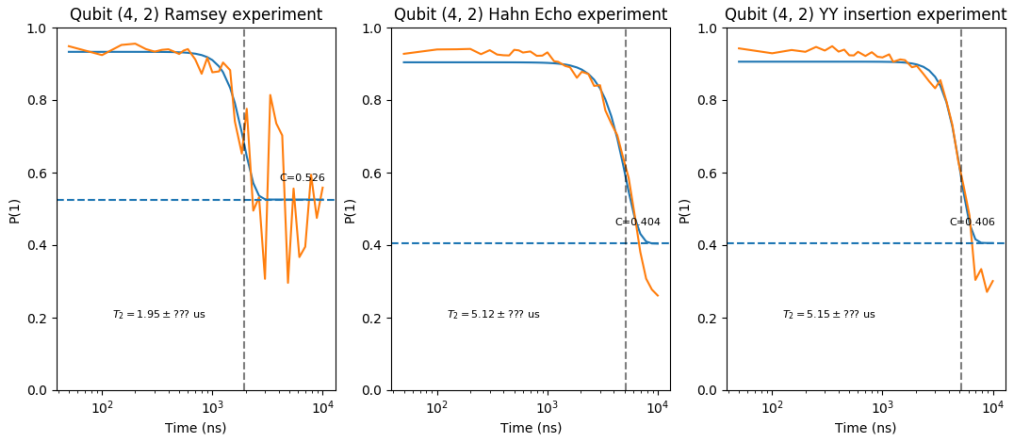


Figure 3.1: This T_2 diagnostic battery for qubit $(4, 2)$ using 2000 repetitions shows (left) unmodified Ramsey experiment, (middle) a Hahn echo T_2 experiment, and (right) insertion of as many Y gates as possible between preparation of $|+\rangle$ and readout of $|1\rangle$. The third experiment resembles a CPMG pulse sequence with a wait duration of $\delta t = 25$ ns, and its similarity to the Hahn echo outcomes suggests that low frequency noise is the dominant contribution to dephasing in the system.

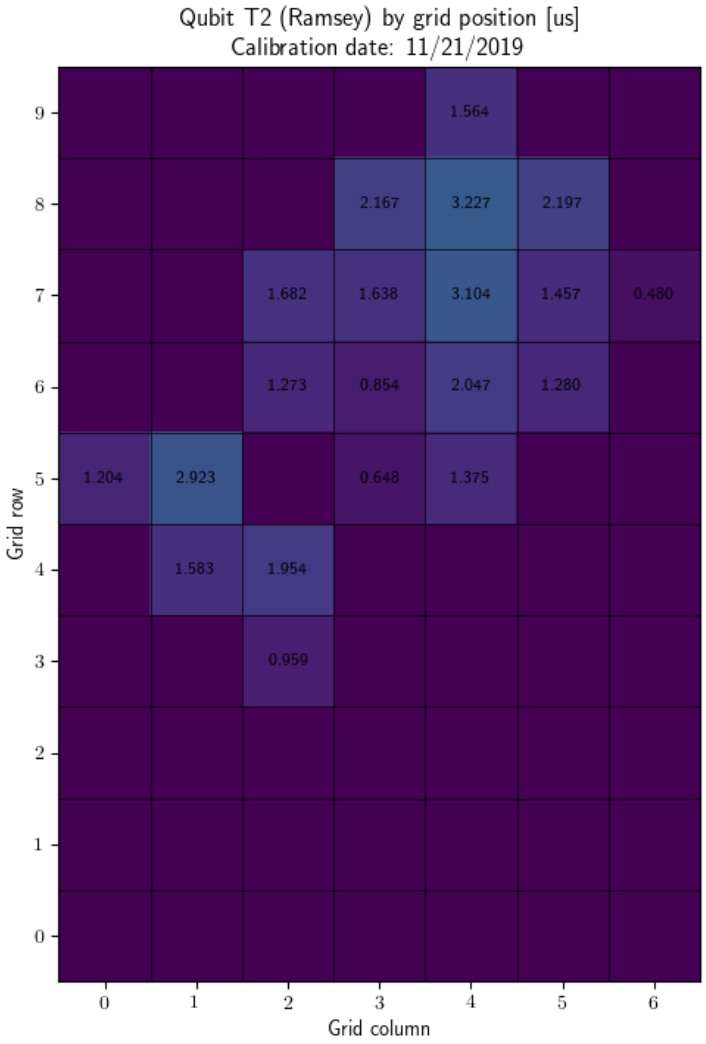


Figure 3.2: T_2 values (shaded according to T_1 values over the same qubit set) extracted from Ramsey T_2 experiments using Equation 3.1 are in the 1-2 μs range. Empty squares indicate failed fits due primarily to state preparation error.

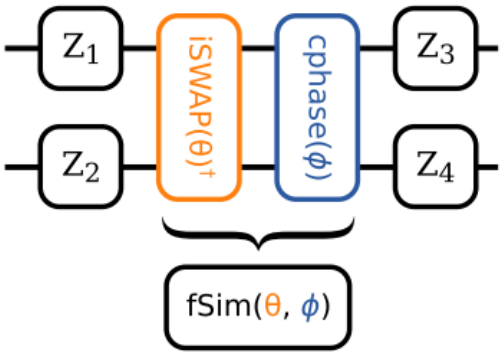


Figure 3.3: The unitary model for implementing $\sqrt{\text{iSWAP}}$ ($\theta = 45$, $\phi = 0$) on hardware [3]. Each two qubit gate on the superconducting qubit processor is initially accompanied by unintended relative phasing of qubits that can be gathered into local rotations before and after the intended two-qubit gate. Hardware calibrations are configured to optimize single-qubit phase gates to cancel these unitary errors.

3.2 Mitigating crosstalk error

"Crosstalk" refers to decoherence or unitary error that occurs due to physical coupling between qubits or control signals when two or more quantum gates are applied simultaneously. In the following sections, I primarily focus on optimizations/diagnostics based on the unitary model for $\sqrt{\text{iSWAP}}$ described in [3] and depicted in Figure 3.3.

3.2.1 Optimization over gate simultaneity graphs

Operating on the knowledge that gates exhibit crosstalk errors, there is an inherent tradeoff between the fidelity that is gained by running gates sequentially when they might otherwise be run simultaneously, versus the additional decoherence that results from the additionally circuit depth that is incurred by running gates sequentially. Given some circuit fidelity metric, this tradeoff can be cast into an optimization problem by parametrizing the degree of gate simultaneity (inversely proportional to some function of depth) in the circuit.

Given a set of operations acting on disjoint sets of qubits P, let the "simultaneity graph" be $\mathcal{G}_s(P) = (E, V)$ where edges indicate simultaneity of operations and vertices represent operations on fixed qubits, and let $G_s(n, P) = \{\mathcal{G}_s(P) : |V| = n\}$ be the set of all simultaneity graphs over n qubits. Then the degree of simultaneity of a circuit

implementing all operations in P is *literally* the degree of the simultaneity graph, and the existence of many graph implementations of a given degree demonstrates the different schemes of parallelization available over P.

As will be seen later, this non-uniqueness can be exploited to discover pairs of operations that are more prone to crosstalk without having any prior knowledge of the physical implementation of the operations that leads to crosstalk. In other words, this diagnostic is hardware-agnostic and doesn't require notions about proximity of qubits, proximity of microwave control lines, the existence of defects supporting stray capacitances/inductances between disconnected components, etc.

I implemented two instances of simultaneity graphs over 12-qubit circuits (graphs shown in Figure 3.4), using the Hellinger distance between output hardware distributions with respect to noiseless simulation as a fidelity metric. For each element of $\mathcal{G}_s(P) \in G_s(12, P)$, I rearranged all entangling operations in an instance of a kernel circuit to respect the simultaneity imposed by $\mathcal{G}_s(P)$, and modified the circuit to use only the entangling gates present in P to control for which gates were being analyzed. Figure 3.5 shows two instances of kernel circuits that have been modified in this way. I ran 5000 repetitions per circuit, repeated for two sets of six entangling gates (therefore probing all gates that would be used in the implementation of a 12-qubit circuit), for each of eight different kernel circuits, for a total of 1.1 million circuit repetitions.

Figure 3.6 shows the results of the simultaneity graph diagnostic. There is a clear trend towards improved fidelity (lower Hellinger distance with respect to the simulated model) over all instances of circuits that ran, and accounting for all different simultaneity graphs of a given degree that were implemented. While a few cases indicate that the maximally parallelized operations perform slightly worse than *some* of the degree-3 graphs, the overall trend indicates that any performance gains due to the reduction of crosstalk effects from operations run sequentially is overshadowed by the loss of fidelity due to the increased circuit depth.

3.2.2 Optimization of unitary model corrections from convergence of random circuits

A hypothesized method of detecting consistent unitary errors of the type depicted in Figure 3.3 was by analyzing the difference in a random circuit's behavior in the presence of such errors, versus its behavior without consistent control errors. This method is appealing because the behavior of random circuits has been studied extensively theoretically [33, 8]

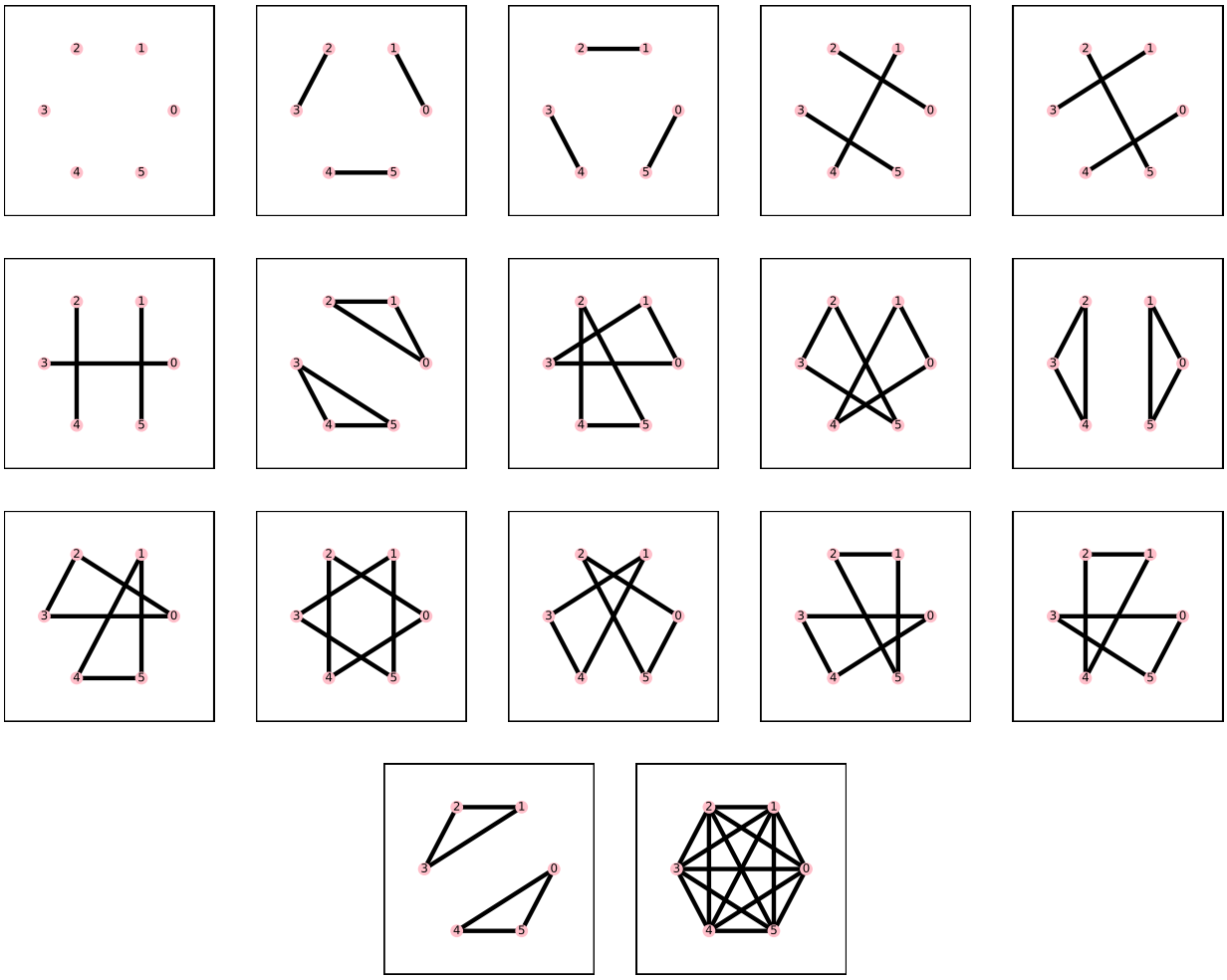


Figure 3.4: All simultaneity graphs $\mathcal{G}_s(P)$ over six two-qubit gates $G_s(12, P)$.

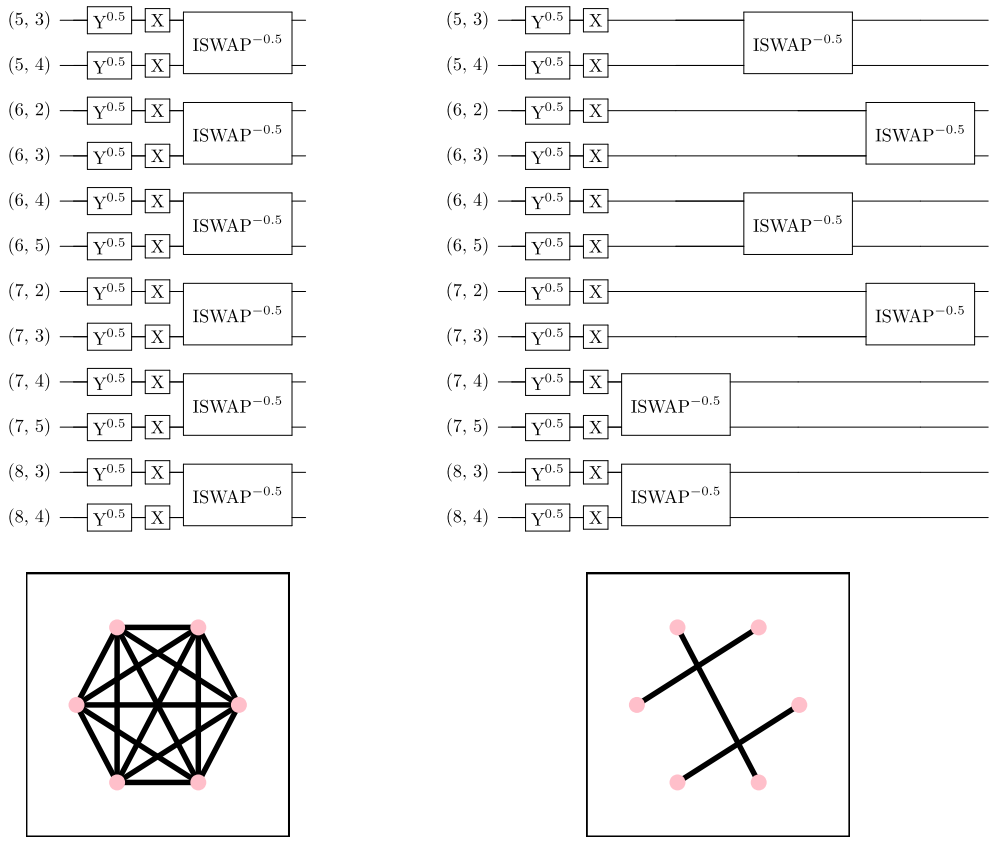


Figure 3.5: Sample of correspondence between simultaneity graphs and circuit configurations.

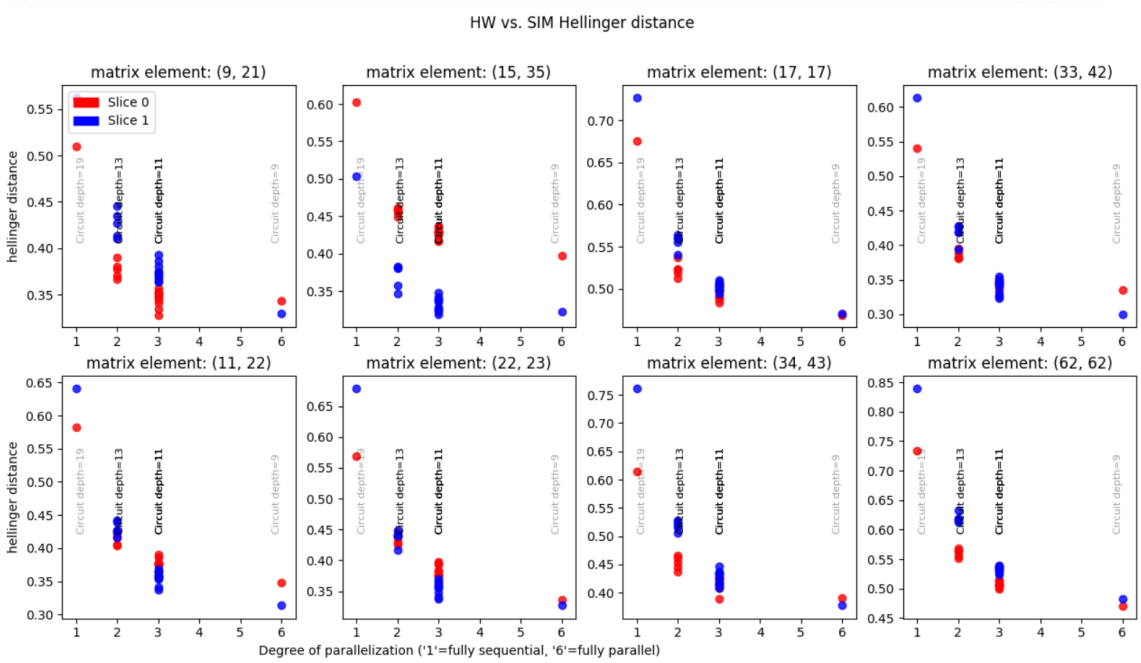


Figure 3.6: Hardware results demonstrate a clear improvement in performance for maximally parallelized two-qubit gates, meaning that crosstalk error is outweighed by decoherence due to increased idle time for executing two-qubit gates sequentially. Each plot corresponds to a different matrix element; the horizontal axis plots the degree of one of the 14 graphs taken from Figure 3.5 (the larger the degree, the more parallel the execution of gates) while the vertical axis plots the Hellinger distance between distributions taken from raw hardware results vs. noiseless simulation. Different colors correspond to different sets of two-qubit gates that can be run fully parallel.

and have been validated experimentally on hardware similar to that used for this work's experiments [3].

The Porter Thomas distribution [8] describes the exponential distribution of counts of bitstring outcomes observed when measuring a state that has been sampled Haar-randomly from Hilbert space. That is, if N is the the total number of observable outcomes (exponential in the number of system qubits) and p describes the the probability of a given bitstring, then the distribution of the probabilities of bitstrings *frequencies*[2] is expected to be:

$$P(Np) \propto \exp(-Np) \tag{3.2}$$

Convergence to this distribution is a function of circuit depth and number of qubits in the random circuit being sampled. With this in mind, a diagonstic experiment for detecting $\sqrt{\text{iSWAP}}$ unitary errors via convergence to the Porter Thomas distribution is:

1. Simulate a noiseless random circuit (called RC1) composed of randomly selected single qubit gates and $\sqrt{\text{iSWAP}}$ gates
2. Run a modified circuit (RC2) that is identical to RC1 except for parametrized single-qubit phase correction gates of the kind shown in Figure 3.3 on hardware
3. Compare each circuit's distribution of bitstring frequencies to Equation 3.2, for example using KL divergence
4. Modify the parameters in RC2 to match more closely the convergence behavior of RC1, thereby nullifying any additional convergence provided by hidden phase errors attached to each two qubit gate

This method requires demonstrating that a given circuit architecture (i) converges to the distribution in Equation 3.2 and (ii) does so in a quantitatively different manner than a similar circuit. Figures 3.7-3.8 demonstrate that these conditions are not met, and so this method was not pursued on actual hardware.

3.2.3 Optimization of unitary model corrections from Quantum Process Tomography

The results of Section 3.2.1 indicate that maximally parallelizing operations improves performance over any partially sequential alternative, but does not rule out the possibility that

[2] In other words, the histogram of the populations in a histogram of bitstring outcomes - a *metahistogram*!

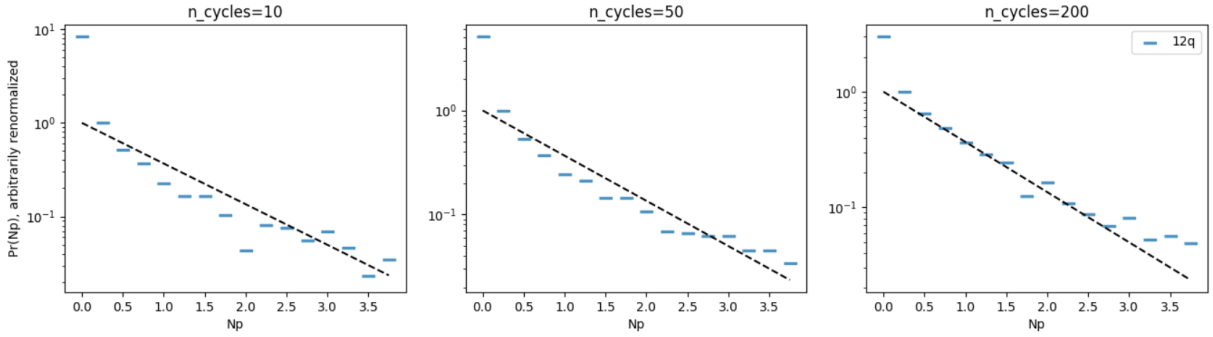


Figure 3.7: Convergence of a 12 qubit random circuit to the Porter Thomas distribution (Equation 3.2) for different numbers of "cycles" (each cycle consisting of a layer of single qubit gates followed by a sparse layer of two qubit gates). The dashed line shows Equation 3.2, and demonstrates that (qualitative) agreement with the desired distribution does not occur until some depth greater than 50 cycles.

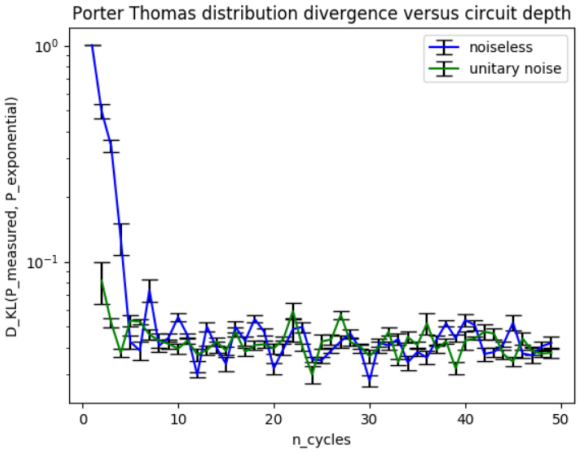


Figure 3.8: Convergence of random circuits in a noiseless setting versus the same random circuits with unitary noise presence, as reflected by KL-divergence of the observed distribution with respect to Equation 3.2. For each circuit depth, 5 different simulated random circuits were generated and sampled for 150,000 repetitions to produce a measured distribution $P_{measured}$, from which $D_{KL}(P_{measured}, P_{exponential})$ was computed (error bars represent standard error in the five computed values). There is no statistically significant difference in the convergence behaviors of these circuits beyond $5 cycles$, which is far too shallow to reach Porter Thomas distribution as seen in Figure 3.7.

crosstalk-related decoherence arises in the fully-parallel implementations. As described in Section 2.3.4, gate infidelity can be attributed to a combination of unitary and non-unitary errors. Given a compelling model for parameterized gates that would lead to unitary errors and an acceptable single-qubit gate fidelity then one can attempt to optimize parameters in this model, thereby learning corrections that will nullify the unitary errors introduced by crosstalk.

More specifically, suppose a two-qubit gate is represented by the unitary U, and the two-qubit gate infidelity in the hardware as the noise channel is represented as $\mathcal{D}$, a combination of unitary and non-unitary effects. The goal is to implement a parameterized unitary $V(\vec{\theta})$ and optimize its parameters to find

$$\text{argmin}_\theta \left(f \left(\mathcal{D}_{V(\vec{\theta})}, U \right) \right) \tag{3.3}$$

where f is some fidelity metric between unitaries. For this experiment, I implemented the unitary model for $\sqrt{\text{iSWAP}}$ described in [3] (Figure 3.3). However, the fidelity metric from that work isn't relevant here since that metric was specific to circuits approaching random unitaries. Instead, I conduct QPT (Section 2.2.1) to determine the CPTP Choi matrix representation of the entangling gates when run in a maximally parallel context [26, 37]. After preparing a full tomography experiment in CIRQ, I used the optimization algorithm from [26] provided in the JULIA package *Schrodinger.jl* [4] to analyze the experimental results.

Simulated QPT optimization results

This section demonstrates efficacy of an optimizer to solve for an optimum according to Equation 3.3 with respect to a set of hidden local phases as shown in Figure 3.3, namely

$$V(\theta) = Z_0^b Z_1^c \sqrt{\text{iSWAP}} Z_0^a \tag{3.4}$$

The key observation is that the Choi representation $J(\mathcal{D}_{V(\theta)})$ for a (unitary) channel enacting Equation 3.4 (i.e. $\mathcal{D}_{V(\theta)} : \rho \to V(\theta)\rho V^\dagger(\theta)$) admits an *exact* solution for the parameters a, b, c. For example, element (0, 5) of the matrix form of $J_V \equiv J(\mathcal{D}_{V(\theta)})$ can be solved for exactly:

$$(J_V)_{05} = \frac{1}{\sqrt{2}} \exp(-i\pi c) \tag{3.5}$$

which immediately yields the solution:

$$c = \frac{i}{\pi} \ln\left(\sqrt{2}(J_V)_{05}\right) \tag{3.6}$$

Repeating this process for a and b returns an overconstrained system of 9 equations with three unknowns. In practice, statistical uncertainty in the elements of $J(\mathcal{D}_{V(\theta)})$ makes constrained simultaneous equation solving impossible, and unincorporated noise effects will invalidate the exact solutions for parameters. Instead, the exact solutions for a, b, c of the form in Equation 3.6 are used as initial estimates for the $\vec{\theta}$ minimizing Equation 3.3, which is then fed into a gradient-free optimizer.

Running this procedure with a Nelder Mead optimizer and a unitary error model like Equation 3.4 resulted in 99.9999% accurate recovery of a variety of random input parameters a, b, c, demonstrating that in a noiseless simulation setting this unitary error model for $\sqrt{\text{iSWAP}}$ can be perfectly corrected for. However in the following section it is shown that this result could not be extended to real hardware outcomes, based solely on the uncertainty introduced by bitflip errors.

Hardware QPT results

The goal of the experiment is to compare the fidelity of a $\sqrt{\text{iSWAP}}$ gate executed in isolation to the fidelity when the same gate is executed with other gates in parallel. In addition, since single qubit gate errors and $T1/T2$ decoherence introduce error into the QPT experiment that is unrelated to the two qubit gate fidelity, for any given qubit pair I also ran QPT on an identity gate, which is implemented by telling the processor to "wait" for 32 ns (the duration of a $\sqrt{\text{iSWAP}}$ gate). The three circuits for implementing this experiment are shown in Figure 3.9.

In addition to running a control experiment, I also implemented postprocessing for bitflip correction (see Section 5.3.2) which typically provided around a 10% improvement to observed gate fidelities. Table 3.2.3 summarizes results from performing QPT using the circuits from Figure 3.9 and applying bitflip correction, which propagates roughly $1/\sqrt{5000} \approx 2\%$ uncertainty originating in the Poisson uncertainty in the p_0, p_1 values determined from experiments using 5000 repetitions.

The results from Table 3.2.3 are bleak. An idealized interpretation might be to use the Identity fidelities as normalization factors to back out "True" parallel and isolated $\sqrt{\text{iSWAP}}$ fidelities that more closely match the XEB fidelities reported in calibration data.

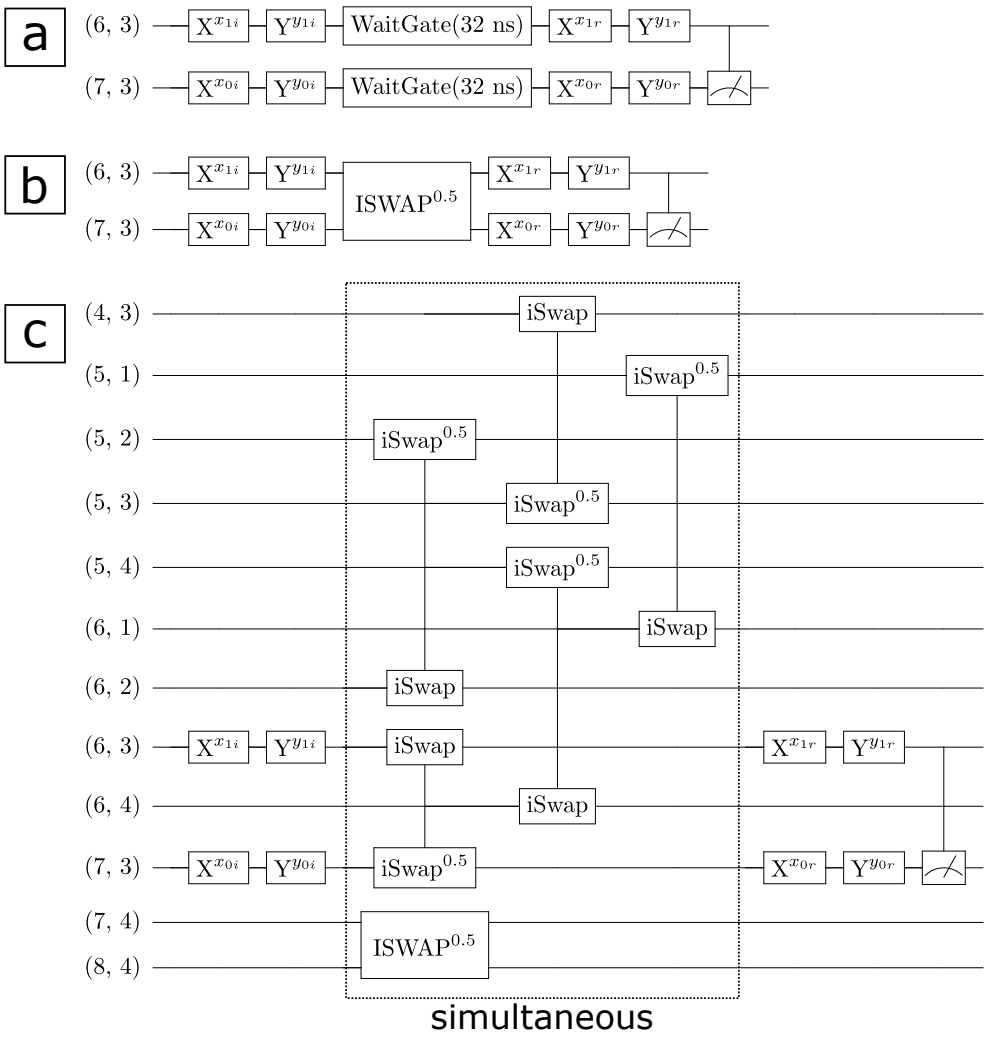


Figure 3.9: (a) Control circuit that performs tomography on an identity gate to establish an upper bound on the expected fidelity observed from a QPT experiment on any given pair of qubits (b) a standard QPT experiment to compute fidelity of an isolated $\sqrt{\text{iSWAP}}$ gate (c) a QPT experiment that computes fidelity of a $\sqrt{\text{iSWAP}}$ gate that is run in parallel with five other two-qubit gates. The circuits are structured and parameterized so that varying the angles on each X and Y rotation allows measurement of the POVM $P = \{\frac{1+i}{2}|0\rangle + \frac{1-i}{2}|1\rangle, \frac{1-i}{2}|0\rangle + \frac{1+i}{2}|1\rangle, \frac{1+i}{2}|0\rangle + \frac{1+i}{2}|1\rangle, \frac{1-i}{2}|0\rangle + \frac{-1+i}{2}|1\rangle, |0\rangle, |1\rangle\}$ as well as preparation of all states of the form $|A\rangle|B\rangle$ for $|A\rangle, |B\rangle \in P$.

Table 3.1: Fidelities taken from XEB calibration data and results from QPT experiments run on the circuits from Figure 3.9. QPT fidelities (f_{QPT}, Equation 2.7) are reported to two significant figures due to the 2% uncertainty introduced into the bitflip correction algorithm.

Qubit pair:	(7, 3)/(6, 3)	(6, 3)/(6, 4)	(5, 2)/(6, 2)	(6, 1)/(6, 2)
$\sqrt{\text{ISWAP}}$ f_{XEB} (%)	99.1	98.3	99.3	98.9
Identity f_{QPT} (%)	82	82	95	94
Isolated $\sqrt{\text{ISWAP}}$ f_{QPT} (%)	83	81	96	94
Parallel $\sqrt{\text{ISWAP}}$ f_{QPT} (%)	82	82	96	94

For example, on the (6, 1)/(6, 2) qubit pair (using an additional insignificant digit not presented in Table 3.2.3):

$$\text{“True” isolated } \sqrt{\text{ISWAP}} \text{ fidelity} \approx 0.944/0.948 = 99.6\% \tag{3.7}$$

$$\text{“True” parallel } \sqrt{\text{ISWAP}} \text{ fidelity} \approx 0.942/0.948 = 99.3\% \tag{3.8}$$

However, this treatment doesn't make sense when applied to some of the other qubit pairs. The more general (and correct) interpretation is that there is no statistically significant method of confirming XEB fidelities against the process tomography results. Therefore, this application of standard quantum process tomography cannot be used to verify the existence of parallel $\sqrt{\text{ISWAP}}$ errors to a degree of uncertainty less than the 2% error in reported bitflip probabilities. For the same reason, there would be no way of confirming the success of a hardware optimization routine like the one described in the previous section. A couple of possible solutions to this limitation are:

- Conducting a “self-consistent” process tomography experiment [32]: This scheme uses fine-grained control of single qubit gates to construct tomography experiments where the process matrices for the POVM measurement and state preparation gates are known in advance, thereby mitigating error introduced to reconstruction of two-qubit gates due to SPAM errors.
- Better statistics for p_0, p_1 values: The uncertainty in the gate fidelities are propagated directly from the initial uncertainty in bitflip probabilities for state readout. Reducing these uncertainties might allow more conclusive comparison between fidelities for $\sqrt{\text{iSWAP}}$ gates run in isolation versus parallel.

3.2.4 Phase refocusing noise mitigation

The improvement of T_2 values observed in the Hahn echo experiment compared to the standard Ramsey tomography suggests that phase coherence of the qubits can be preserved by refocusing (executing "echos") during idle time in the circuit. In practice this corresponds to running pairs of Y gates (duration=25 ns each) during each 50 ns time period over which a qubit is left untouched.

Figure 3.10 shows a sample histogram for which spin echo optimization was implemented. The KL-divergence D_{KL} was used compare results from hardware with/without refocusing optimization to different simulations. Unfortunately, detailed analysis of these outcomes is complicated by the depth of the circuit (49) combined with the KL-divergence's status as a pseudo-metric in probability space (D_{KL} does not obey the triangle inequality) so that its hard to reliably quantify convergence of the optimized hardware distributions towards/away from simulated outcomes.

The qualitative trend in D_{KL} values comparing the hardware outcomes for circuits with and without refocusing suggests that this hardware optimization does not affect the performance of the hardware much. Experimental results from the optimization applied to a larger set of shallower circuits could be used to better determine efficacy of this optimization.

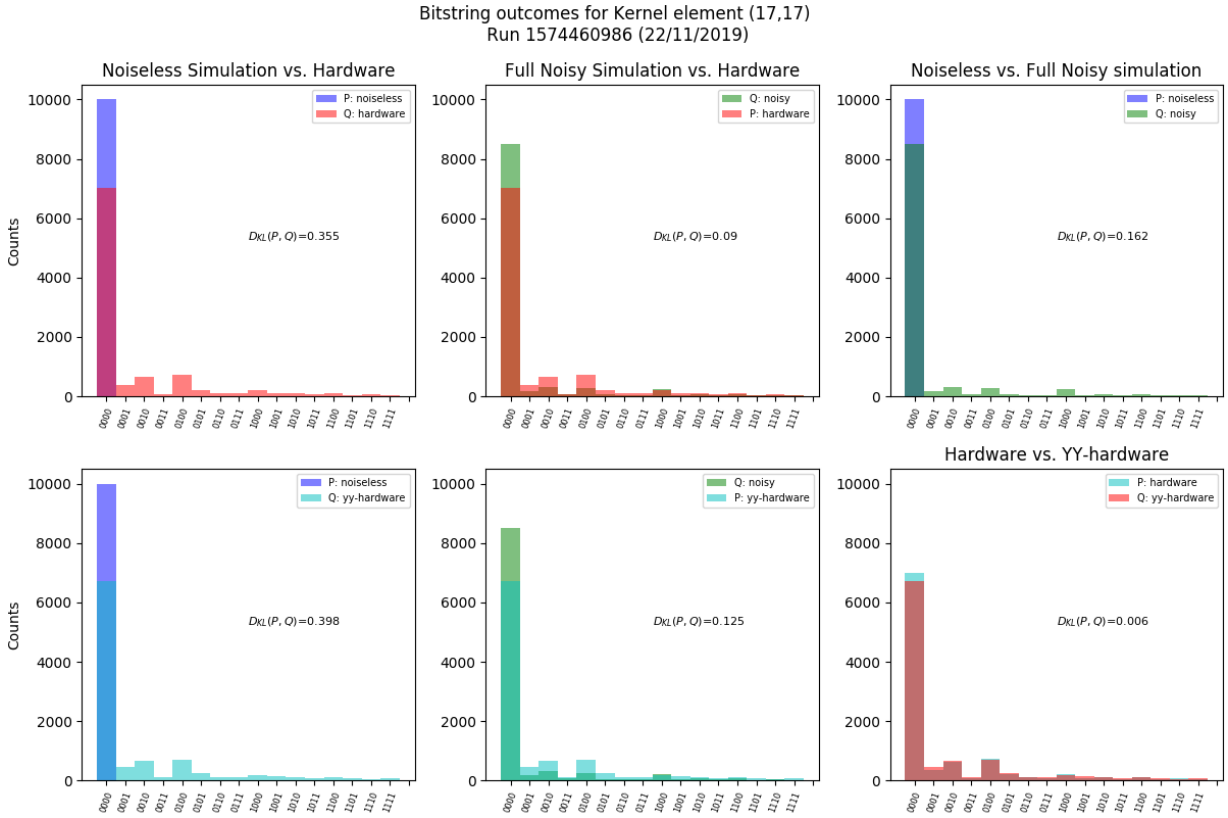


Figure 3.10: For the given circuit (i.e. implementing a depth-49 identity gate) on $n = 4$ qubits, there is little evidence for performance enhancement through the use of refocusing. This is especially clear for the comparison of hardware with/without refocusing optimization (far bottom right plot) in which the two distributions are nearly identical.

Chapter 4

Theory of kernel method classifiers

This chapter introduces the Support Vector Machine, a supervised learning algorithm that uses kernel functions to learn about data.

4.1 Supervised Learning

Supervised learning algorithms are tasked with the following problem: Given a *dataset* $\mathcal{X} \subset \mathbb{R}^d$ composed of n-dimensional datapoints and the corresponding classes $\mathcal{Y} \subset \mathbb{R}$ that each datapoint belongs to, construct a function f that can successfully predict $f(x_i) = y_i$ given a datapoint-label pair taken from the dataset $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$.

In the process of constructing a classification function f, supervised learning algorithms need to have some way of determining whether the classifier's predictions match reality. This relationship is usually measured using a cost function, whose value depends on how well the function f predicts the class y_i corresponding to each $x_i \in \mathcal{X}$, while also enforcing that f is well behaved. For a size-n dataset ($|\mathcal{X}| = n$) a typical cost function might take the form:

$$\begin{aligned} c &: (\mathcal{X} \times \mathbb{R}^2)^n \to \mathbb{R} \\ &= c((x_1, y_1, f(x_1)), ...(x_n, y_n, f(x_n))) \end{aligned} \tag{4.1}$$

Once a cost function is chosen, *training* a supervised learning algorithm consists of having some guess for the initial form of f, providing it with a subset of data points to compute $f(x_i)$, comparing each outcome to y_i via the cost function, and then adjusting f

in some way to improve this computed loss (and therefore bring $f(x_i)$ closer to y_i over the entire dataset). Since the algorithm is exposed to only a subset of possible input data, it may bias f to perform well for that specific subset of data without capturing trends over the broader population. If this happens, the model will fail to generalize its predictions, and so *testing* the algorithm refers to the process of exposing the trained algorithm to previously unseen datapoints to measure its generalizability.

The last important detail remaining is how to structure f so that it can perform well as a classification function. Since the space of functions on d-dimensional data is infinitely large, one challenge for supervised learning algorithms (and in machine learning in general) is to provide a form for f that both generalizes well to a broad class of data inputs and is capable of achieving optimal performance for any given dataset. The next section will introduce the methods associated with constructing one such optimal classifier.

4.2 Kernel methods

The main focus of this work is to use quantum hardware to efficiently find similarities between data points $x_i \in \mathcal{X}$. A kernel function is a tool for computing some degree of similarity between points, and is defined as a map $k : \mathcal{X} \otimes \mathcal{X} \to \mathbb{R}$ that satisfies the following condition for positive definiteness:

$$\sum_{x_i, x_j \in \mathcal{X}} c_i^* c_j k(x_i, x_j) \geq 0 \tag{4.2}$$

for all possible choices $c_i \in \mathbb{C}$. For finite $\mathcal{X}$, k will take on exactly $|\mathcal{X}|$-many values, the two-dimensional array of which is referred to as the Gram matrix:

$$K_{ij} \equiv k(x_i, x_j) \tag{4.3}$$

The kernel is symmetric, and therefore requires only $n(n+1)/2$ computations of $k(x_i, x_j)$ to describe an $n \times n$ Gram matrix, though the general cost of computing the Gram matrix scales as $\mathcal{O}(n^2)$). Furthermore, $k(x_i, x_i) \equiv 1$ due to the kernel function's status as an inner product (see below); here and in future sections the term "diagonal" will refer to kernel elements like $k(x_i, x_i)$ and "off-diagonal" will refer to elements like $k(x_i, x_j)$ when $i \neq j$.

In addition to these basic requirements, the kernel admits two important properties that make it a useful tool for classifying data:

- **The existence of a corresponding Hilbert space** [2]: Every function satisfying the positive definite requirement of Equation 4.2 has a corresponding Hilbert space $H(\mathcal{X})$ and mapping $\phi : \mathcal{X} \to H(\mathcal{X})$ such that

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{H(\mathcal{X})} \tag{4.4}$$

  where $\langle \cdot, \cdot \rangle_{H(\mathcal{X})}$ is the inner product corresponding to the Hilbert space $H(\mathcal{X})$. In other words, every kernel function is understood to be the inner product of mappings of input data into *some* Hilbert space, even without specific knowledge of what space or mappings satisfy the relation in Equation 4.4.

- **The representer theorem** [53, 48]: Any classifier that minimizes a typical cost function like Equation 4.1 over datum-label pairs in $\mathcal{X} \times \mathcal{Y}$ admits a representation of the form

$$f(x) = \sum_{i=1}^{|\mathcal{X}|} \alpha_i k(x, x_i) \tag{4.5}$$

  The key point is that any new element taken from $\mathcal{X}$ can be optimally classified (with respect to a given kernel k) using *only the elements in the complete Gram matrix over* $\mathcal{X}$. In other words, a classifier that is trained using a kernel function or Gram matrix on a dataset of size n needs at most n free parameters to achieve optimal classification with respect to some loss that compares $f(x_i)$ to y_i for all i. This is offers great reduction in complexity in the search among infinite classifier functions of the form $f : \mathbb{R}^d \to \mathbb{R}$.

As mentioned in Chapter 1, a conceptual way of justifying the use of a kernel is that it implicitly maps data points into a higher dimensional space in which the points may be successfully classified by a linear classifier. The results of [2] provide rigorous justification for this view via the association of each positive definite kernel with an inner product on some Hilbert space, and the representer theorem motivates the use of classifiers that digest kernel functions (since such functions are strictly upper bounded in complexity). With the above features of kernels established, Section 4.2.1 will develop the formalism for a classifier that can be trained on known values of $k(x, x')$.

4.2.1 Support Vector Machine formalism

This section describes Support Vector Machine classifiers following a combination of the derivations of [34] and [52].

Geometric problem statement for the SVM

A basic Support Vector Machine (SVM) seeks to find a $d-1$-dimensional hyperplane that separates points in a dataset according to their labels. The dataset is defined as a size-n subset of d-dimensional vectors, $\mathcal{X} \subset \mathbb{R}^d, |\mathcal{X}| = n$, and a datapoint $x_i \in \mathcal{X}$ will have a corresponding label $y_i \in \{-1, 1\}$. The classification task will be the most accurate if the hyperplane is situated as far as possible from points in each class (maximizing the "margin" - see Figure 4.1). The margin is determined by the distance of the closest datapoint to the dividing hyperplane; defining v_i as the perpendicular distance separating a point x_i from any given choice of hyperplane, the margin is defined by the shortest of these distances:

$$v \equiv \min_i v_i \tag{4.6}$$

A hyperplane in $\mathbb{R}^d$ has the general formula

$$\langle w, x \rangle + b = 0 \tag{4.7}$$

Figure 4.1 shows that the vector describing perpendicular distance of point x_i to the hyperplane (at point $\mathcal{P}_i$) is given by $v_i \frac{w}{||w||}$, which combined with the fact that point $\mathcal{P}_i$ must satisfy Equation 4.7 yields:

$$\langle x_i - v_i \frac{w}{||w||}, w \rangle + b = 0 \rightarrow v_i = \frac{1}{||w||}(\langle w, x_i \rangle + b) \tag{4.8}$$

This distance v_i will be positive or negative depending on what side of the hyperplane x_i falls on, but multiplying by the binary class y_i will enforce that it always takes on a positive value:

$$v_i \rightarrow y_i \frac{1}{||w||}(\langle w, x_i \rangle + b) \tag{4.9}$$

From Equation 4.9, v_i (and therefore v) will be maximized if $||w||$ is minimized: *Maximizing the margin is equivalent to minimizing* $||w||$.

The goal of maximizing v must be constrained by the requirement that the SVM accurately classifies data. For a given choice of hyperplane and v, a given data point x_j that falls well on one side of the hyperplane (by some distance c) will be assigned the same class as all of the other points in that region[1]. For binary classification (i.e. $y \in \{1, -1\}$) this can be stated with a single inequality:

[1]The simplest case of a working SVM assumes that the condition in Equation 4.10 is satisfiable for all $x_i \in \mathcal{X}$ simultaneously. In practice, a more general treatment is used that allows for some error (mixture of classes on either side of the dividing hyperplane) and some degree of infeasibility. See [52].

$$y_i(\langle x_i, w\rangle + b) \geq c \tag{4.10}$$

Comparing to Equation 4.9, a choice of $c \geq ||w||v$ will enforce that all classified datapoints are on the edge of or beyond the margin in Figure 4.1. But observing Equation 4.9, there exists some arbitrary scaling of w and b for which v_i is unchanged but $v_i||w|| = 1$, namely $w, b \rightarrow \frac{w}{||w||v_i}, \frac{b}{||w||v_i}$. This rescaling is allowed because it doesn't affect the goal of the SVM (to minimize v_i), but it allows the choice of $c = 1$ to satisfy that all classified points lie on the edge of the margin or further. This choice of c, Equation 4.10, and the goal of minimizing $||w||$ (to maximize the margin) results in a constrained optimization problem for the SVM:

$$\begin{aligned} &\text{minimize } \frac{1}{2}||w||^2 \\ &\text{subject to } y_i(\langle x_i, w\rangle + b) \geq 1 \end{aligned} \tag{4.11}$$

where minimization of $||w||$ was replaced with minimization of $\frac{1}{2}||w||^2$ for convenience in the following derivation. The purpose of the next section is to recast and solve this constrained optimization problem.

Lagrangian/dual problem statement for the SVM

The constrained optimization problem of Equation 4.11 can be solved using the method of Lagrange multipliers, which states that the extrema of a function $f(x)$ subject to an inequality constraint $g(x) \leq 0$ can be solved by constructing a Lagrangian $\mathcal{L} = f(x) - \alpha g(x)$, enforcing $\alpha \geq 0$, and setting $\boldsymbol{\nabla}\mathcal{L}(x, \alpha) = 0$. Applying this to the SVM problem, the constraint in Equation 4.11 must hold for all $i = 1, \ldots, n$ and so the generalized Lagrangian to optimize is:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\langle w, x_i\rangle + b) - 1\right) \tag{4.12}$$

where the inequality constraint will require that all $\alpha_i \geq 0$. Setting the components of

$\nabla\mathcal{L}$ to zero yields the outcomes:

$$\partial_b \mathcal{L} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{4.13}$$

$$\partial_w \mathcal{L} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \tag{4.14}$$

Then, solving Equation 4.14 for w and substituting into Equation 4.12 gives:

$$\mathcal{L}(w, b, \alpha) \to \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \langle \alpha_j y_j x_j, \alpha_i y_i x_i \rangle - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\sum_{j=1}^{n} \langle \alpha_j y_j x_j, x_j \rangle + b \right) - 1 \right] \tag{4.15}$$

$$= -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_i \alpha_j y_j \langle x_j, x_i \rangle + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} y_i \alpha_i \tag{4.16}$$

Applying the outcome of Equation 4.13 and gathering other constraints, the "dual form" of the SVM optimization problem can thus be stated as:

$$\text{find } \max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_i \alpha_j y_j \langle x_j, x_i \rangle \tag{4.17}$$

$$\text{subject to } \alpha_i \geq 0, \quad i = 1, \ldots, n$$

$$\text{subject to } \sum_{i=1}^{n} \alpha_i y_i = 0$$

With the optimal α chosen, the classifier function assigning a label to an input $x \in \mathcal{X}$ is:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle + b \tag{4.18}$$

Key features of the SVM

This final form of the SVM classifier function simply needs n optimal values for $\alpha_1 \ldots \alpha_n$ to be determined during training by solving Equation 4.17, and by the representer theorem

this optimum is guaranteed to exist for f. Furthermore, the number of nonzero α_i is typically small and corresponds exactly to the number points used to define the margin of the dividing hyperplane [34]. These points were referred to as "support vectors" in the original literature, hence the name of the Support Vector Machine.

The key insight for constructing SVMs capable of separating non-linearly separable datasets is that both the optimization problem in Equation 4.17 and the corresponding classifier function of Equation 4.18 do not depend explicitly on elements of $\mathcal{X}$, only on the positive definite, symmetric inner product on $\mathbb{R}^d$. By substituting a kernel function $k(x_i, x_j) \equiv \langle \phi(x_i), \phi(x_j) \rangle$ (recall that the kernel function *always* has an associated Hilbert space for which it is an inner product) for $\langle x_i, x_j \rangle$, the SVM becomes sensitive to relationships in the mapped data $\phi(x_i)$ *without the need to explicitly compute these mappings*, and capable of performing classification with nonlinear boundaries using Equation 4.18. This is a key mechanism of the SVM: **Linear class boundaries on high-dimensional mappings of data correspond to highly nonlinear boundaries in the space of the data.**

The substitution of kernel functions for inner products in the space of data is known as the "kernel trick" [9] and is the key to building powerful SVM classifiers. It is also essential for using the quantum Hilbert space and its associated kernel for performing useful classification, since the explicit mapping of data into quantum state space $\phi(x_i)$ will require exponential resources to compute in general. The next section will demonstrate how to compute a kernel function using a universal quantum computer, which can then be passed into an SVM (or other more general algorithms) to perform classical classification.

4.3 Quantum kernel methods

This section reviews the formalism for quantum kernel methods as first introduced in [49, 22]. From Section 4.2, an important consequence for any kernel function k is that there exists some Hilbert space $\mathcal{F}$ and a mapping $\phi : \mathcal{X} \to \mathcal{F}$ such that [2]:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \tag{4.19}$$

A quantum kernel just considers a mapping into a complex Hilbert space accomplished via some encoding unitary $U(\cdot)$ that can be realized on a quantum circuit, i.e.

$$\phi(x) = |\psi(x)\rangle\langle\psi(x)| = U(x) \, |0\rangle\langle 0| \, U^\dagger(x) \tag{4.20}$$

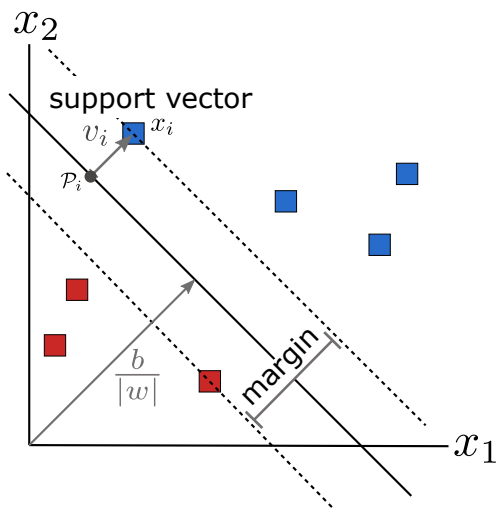


Figure 4.1: The goal of the Support Vector Machine (SVM) is to compute a separating $(d-1)$-dimensional plane that maximizes the "margin" (distance between the dashed lines) between datapoints of different classes in d-dimensional space. Using trigonometry it can be shown that $d \propto \frac{1}{||w||}$, which motivates the minimization of $||w||$

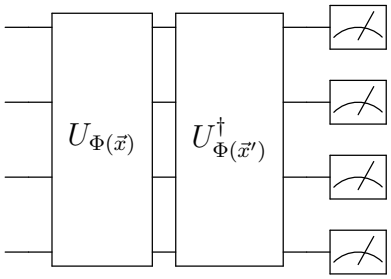
This leads to the quantum kernel to be evaluated as $k(x, x') = \langle \phi(x), \phi(x') \rangle = |\langle \psi(x) | \psi(x') \rangle|^2$. Rephrasing this as an observable with respect to the circuit in Figure 4.25 gives[2]:

$$
\begin{aligned}
k(x, x') &= \langle \phi(x), \phi(x') \rangle && (4.21)\\
&= |\langle \psi(x) | \psi(x') \rangle|^2 && (4.22)\\
&= |\langle 0 | U^\dagger_{\Phi(x')} U_{\Phi(x)} | 0 \rangle|^2 && (4.23)\\
&= P(\underline{0}^n) && (4.24)
\end{aligned}
$$

That is, the kernel k is precisely equal to the probability of observing all zero's at the output (I'll use the notation $\underline{0}$ to represent a *string*, e.g. $\underline{0}^3 = \underline{000}$).

[2]This motivates the use of density matrix formulation for $\phi(x)$, since otherwise the inner product over state vectors $\langle \psi(x) | \psi(x') \rangle$ is not positive in general and the phase of the vectors themselves introduces a problem for the uniqueness of the map ϕ onto real probabilities. See [22].

(4.25)



The finite statistics available to compute $P(\underline{0}^n)$ measured from a circuit U introduces statistical error into the resulting $k(x, x')$. If U is sampled N times to produce N bitstrings, the consequence is that $k(x, x')$ cannot be estimated accurately for $N < 1/|k(x, x')|$. While the distribution of bitstrings sampled from the QKM circuit is unknown in advance, using the uniform distribution over n qubits as a model for typical bitstring frequencies would suggest that $k(x, x')$ could have magnitudes as small as $1/2^{2n}$ when computed on an n-qubit circuit.

However, Section 4.2 introduced the interpretation of a kernel function as measuring a degree of similarity between two points x, x'; if this interpretation is correct and the quantum kernel being computed is useful (i.e. one that does indeed measure similarity between encoded points) then it stands to reason that for x, x' belonging to the same class, sampling $k(x, x') \equiv P(\underline{0}^n)$ would be distinguishable from sampling the uniform distribution. Section 5.6 will explore this behavior for the PROTO-3 quantum kernel specifically, finding sampled kernel values from that architecture are subject to an exponential decrease with respect to the number of qubits in the system.

4.3.1 Encoding functions

In the previous section a datapoint $x \in \mathcal{X}$ was provided as input to the unitary that produces $\phi(x) = |\psi(x)\rangle\langle\psi(x)| = U(x)\,|0\rangle\langle 0|\,U^{\dagger}(x)$, but the kernel circuit still works provided a substitution of the form $x \to f(x)$ for $f : \mathcal{X} \to \mathbb{R}^m$. That is, preprocessing on a datapoint x can arbitrarily reshape and modify the input to U.

One immediate consequence of allowing preprocessing is that the performance of a QKM-assisted algorithm can no longer be solely attributed to the quantum components. A concrete example of this consequence was provided by [29], wherein a deep convolutional neural network processes images into two floats, which are subsequently passed on to a

two-qubit quantum circuit to perform classification. Given that a trivial extension of the classical component of such an architecture could finish the classification, it is questionable whether the quantum part of the model contributed any reasonable degree of performance to the heavy-duty classical encoding process.

To combat the inflation of QKM performance that encoding functions introduce, I followed a guideline similar to the "linear baseline" rule proposed in [55]. In its original form, this rule requires that all pre- and postprocessing be linear so that any nonlinear behavior (supposedly the source of power in a kernel classifier) is attributed to the quantum part of the algorithm. In other words, this forces the quantum kernel function to produce a nonlinearity that allows linear separation in feature space. In a modified form, I impose that no nonlinearities be introduced in encoding functions, but allow for classical non-linear steps to be applied for data postprocessing. This is justifiable because while a nonlinear encoding function fundamentally changes the information content of inputs to the quantum circuit – and therefore modifies the theoretical *algorithmic* performance of the model – the goal of nonlinear postprocessing is to recover information that existed at the output of the quantum circuit but was corrupted by imperfect hardware, thereby modifying the *hardware* performance of the model.

4.3.2 Noise in quantum kernels

Noise contributes another stochastic element to kernel matrices computed on quantum hardware. In the presence of noise, the explicit mapping $x \to |\phi(x)\rangle$ is augmented by non-unitary channels representing decoherence in the system. Taking a simplistic example of a channel $\mathcal{D}$ with Kraus representation $\{M_k\}$ acting on the encoding unitary $U_{\Phi(x)}$ the effective kernel computed is

$$\tilde{k}(x, x') = \mathrm{Tr}(\rho(x)\rho(x')) \tag{4.26}$$

$$= \sum_{ij} \mathrm{Tr}(\langle 0|U^\dagger(x')M_j^\dagger M_i U(x)|0\rangle) \tag{4.27}$$

which effectively replaces the intended data encoding $U_{\Phi(x)}$ with $\mathcal{D}[U_{\Phi(x)}]$, its composition with the noise channel. Non-unitary channels $\mathcal{D}$ will generally fail to preserve inner products over the set of mappings $\{|\phi(x)\rangle\}$ over the input dataset, resulting in the computation of an entirely new kernel $\tilde{k} \neq k$.

4.3.3 On quantum advantage

In the previous sections it was shown that *any* unitary parametrized by datapoints $x_i \in \mathcal{X}$ that can be implemented on a universal quantum computer and has an associated kernel $k(x, x')$ that can be sampled as $P(\underline{0})$.

Primary evidence for quantum advantage of quantum kernel methods relies on demonstrating that there exists some ϕ for which $\langle \phi(x), \phi(x') \rangle$ is classically hard to compute but efficient to compute on a universal quantum computer. Previous work in quantum kernel methods have argued for precisely this advantage of QKMs over classical ones. In [22] it was argued that the hardness of the quantum circuit encoding a kernel function relied on the circuit being "similar" to an algorithm used to solve the hidden shift problem for bent boolean functions of the Maiorana-McFarland kind [47]. An argument of this type requires that the subcircuit $\mathcal{U}_{\Phi(\vec{x})}$ be diagonal in the computational basis (so that it has a classical boolean function equivalent) and therefore does not apply to the modified circuits with more than one layer implemented in this work (see Section 5.2.1). [28] provides another notable mention of justifying QKM circuits strictly from a perspective of classical hardness, wherein the circuit used to produce $k(x, x')$ was chosen based simply on prior evidence that the output of such a circuit corresponds to solutions to classically hard graph problems.

In general, arbitrary circuits of the form $U(x)U^\dagger(x')$ (the exact type that samples from a symmetric p.d. kernel) can be composed at a much quicker pace than proofs for classical (in)tractability of sampling from such a circuit's output, and at the same time offer no intuitive justification for why their corresponding kernel functions would be effective for classifying real data. A secondary, weaker path to motivating the use of QKM is to demonstrate that there exists some quantum feature map ϕ for which a classifier $\hat{f}$ incorporating ϕ outperforms existing classical classifiers. While such a demonstration does not prove any sort of quantum speedup, it allows for the discovery of families of kernel functions that are useful for machine learning regardless of whether these kernels are eventually shown to be classically hard to compute or not. From this perspective, quantum kernel methods can be thought of as a way of conveniently prototyping machine learning algorithms that may have otherwise performed poorly using only well-understood and currently available classical functions.

Quantum kernel methods do not necessarily offer speedup over the computation of popular classical kernels. The RBF kernel $K(\vec{x}, \vec{y}) = \exp(-\gamma||\vec{x} - \vec{y}||^2)$ computed on $x \in \mathbb{R}^p$ has complexity $\mathcal{O}(p)$. Meanwhile classical preprocessing to map data points x into gate parameters requires $\mathcal{O}(p)$ operations, and the circuit architectures used in this work (see Section 5.2) using only nearest-neighbor connectivity has depth $\mathcal{O}(p)$ resulting in a total complexity equal to that of computing the RBF kernel, but with the severe disadvantage of

requiring an interface with quantum hardware. Possible circuits with greater connectivity will then exceed computation of classical kernels in complexity, for example a circuit with all-to-all entangling gates will result in $\mathcal{O}({}_pC_2) = \mathcal{O}(p^2)$.

It is also possible that classical approximations to a given quantum kernel can perform well without the need for quantum hardware. For instance, the method of "Random Kitchen Sinks" (RKS) developed in [43, 45, 44] describes how ensembles of practically random feature space projections can be used to efficiently approximate classical kernels that are otherwise expensive to compute. For circuits with *more than one layer* implemented in this work, the methods of [43] for approximating shift-invariant kernels do not apply since layers of the form $U(x) = \prod_k \exp(iZ_k f(x))$ are trivially non-commuting with interspersed layers of single-qubit Hadamards.

4.3.4 The "curse of dimensionality"

One underlying assumption motivating the use of QKM for machine learning is that the exponentially large Hilbert space describing quantum states can be leveraged to provide greater *expressibility* for classical data by mapping low-dimensional classical datapoints into a higher-dimensional feature space. This section will review a classical example of where this intuition fails in the case of purely classical high dimension feature spaces.

The "curse of dimensionality" in the context of classical machine learning refers to the relationship between the dimensionality of the encoded feature space and the size of the dataset N resulting in a very low density of datapoints. Consider a classifier that assigns the class of a new datapoint x based on the predominant class of previously seen datapoints in some region $R(x) = \{x_i\}$ around x:

$$\hat{f}(x) = \frac{1}{M} \sum_{x_i \in R(x)}^{M} y_i$$

A common version of this selection rule is the K-nearest-neighbor algorithm, in which $R(x)$ is defined as the region containing the K closest training points to x with respect to a metric $d(x, x')$. The performance of this algorithm in the limit as $N \to \infty$ is guaranteed to minimize empirical risk if [18]:

$$\lim_{N\to\infty} K(N) = \infty \tag{4.28}$$

$$\lim_{N\to\infty} K(N)/N = 0 \tag{4.29}$$

That is, the number of neighbors used to classify x must grow with the size of the dataset, but must remain smaller than the dataset itself so that the k-nearest-neighborhood remains nontrivial. (4.28) ensures that the K-NN retains some flexibility with respect to a growing body of observations, while (4.29) implies that for a fixed dimension p, the size of $R(x) \to 0$ as $N \to \infty$ and therefore guarantees that x is classified according to the classes of a neighborhood of points that are *nearly identical to* x.

With the requirements for this model in place, the curse of dimensionality effect is typically demonstrated in two ways:

1. The size of the k-nearest-neighborhood scales like $N^{1/p}$ for N points uniformy distributed in $\mathbb{R}^p$ [18]:
$$d(p, N) = 2\left(\frac{p\Gamma(p/2)}{2\pi^{p/2}N}\right)^{1/p} \tag{4.30}$$
For large p the size of $R(x)$ decreases slowly with respect to N, which means that the k-nearest-neighborhood remains large and the limit (4.29) will be difficult to satisfy.

2. A more heuristic approach considers the vanishing volume of the p-sphere with respect to the p-cube:
$$\lim_{p\to\infty} \frac{V(S^p)}{V(\text{p-cube})} = \lim_{p\to\infty} \frac{\pi^{p/2}}{p\Gamma(p/2)} = 0 \tag{4.31}$$
The reasoning here is that $R(x)$ defined with respect to the 2-norm will be a p-sphere centered at x. If data are uniformly distributed within an n-cube centered at x, then the for large p the fraction of points contained in $R(x) \to 0$. This means that $K(N)$ grows much more slowly, and that satisfying the limit (4.28) becomes substantially more difficult.

Extending these arguments against high-dimensional (classical) feature space to the quantum case is nontrivial, but the arguments should certainly temper the belief that the high dimensional quantum state space will necessarily provide advantages to the classification power of classical machine learning algorithms.

Chapter 5

Computing quantum kernels on superconducting qubit hardware

This chapter details results of preparing, running, and analyzing a full QKM algorithm.

Section 5.1 includes analysis and data manipulation used to prepare the input data to be analyzed using a quantum circuit. Section 3.2.1 describes hardware calibrations that were performed to optimize the performance of quantum circuits in general that do not affect the circuit model representation of the kernel circuits, while Section 5.2 describes specific changes made to the circuit model of the QKM algorithm to improve performance. Section 5.3 describes modifications to the bitstring readouts from the quantum circuit to improve the performance of the QKM-SVM model. Finally, Section 5.5 details the analysis of the SVM classifier trained using quantum hardware

5.1 Data preprocessing

5.1.1 Data compression

All of the classification using real data in this work is based on the simulated Strong Lensing (SL) dataset provided by the Bologna Lens Factory [1]. In this dataset, compressed images are classified as either "lensed" or "not lensed" depending on the presence of observable gravitational lensing.

Without processing, a typical image from this dataset is 101×101 pixels, resulting in datapoints that are $10,201$-dimensional (i.e. $\mathcal{X} \subset \mathbb{R}^{10201}$) and contain too many parameters

to fit into a near term circuit. To rectify this situation, the data was compressed to a number of floats equal to the number of qubits in each circuit using Principle Component Analysis [39, 56]. This compression acts by performing a change of basis on elements of a dataset for which magnitudes of components in the new basis better reflect independent variables explaining trends in the dataset. This has the effect of reducing the number of dimensions required to explain variance in the positions of datapoints, and also assigns a greater importance for explaining this variance to a small number of dimensions in the new basis.

Initial exploration with classical binary classifiers on a variety of compressed data formats ($\mathcal{X} \subset \mathbb{R}^4$, $\mathcal{X} \subset \mathbb{R}^6$, etc.) showed that a reasonable model performance should be somewhere in the 70%-80% range.

5.1.2 Dataset engineering and cross-validation

To make generalizable claims about the performance of a QKM SVM it was important to select a train dataset for which the performance of a trained SVM was highly likely to generalize to new data points. In a hardware context this was especially important because the number of circuit computations needed to compute a kernel matrix on a data set of size N scales like N^2, while the available number of hardware runs was fixed. As a result, computing additional circuits would require a reduction in the number of runs (and therefore increase in variance) used to compute each circuit.

An algorithm to determine the minimum acceptable size of a kernel matrix that produces validated model performance needs to have the following features:

1. Access to a large number of training points - this ensures that the model is not "overfitting" according to idiosyncrasies in the feature-label relationships for a very small dataset

2. Random selection of validation subsets - The model should not be allowed to "choose" as subset that is especially high performance, as this would produce the conditions of (1)

3. Low variance in validated accuracy - model validation techniques offer insights into generalizability of a model, but are equally susceptible to statistical anomalies that may over/under-represent model performance on a small dataset. Therefore, a series of validation runs is required so that the mean of the observed validation scores approaches a true mean generalizable to all data subsets of a fixed size (i.e. "law of large numbers")

Algorithm 1 presents a validation procedure satisfying the above criteria for engineering a dataset that results in generalizable performance with respect to a QKM SVM. In summary, the algorithm determines how reproducible training results from any given subset size are by repeatedly sampling subsets of that size and testing models trained from each subset on the subset's own members. One motivating factor for employing this algorithm is that it can be run many times on a single Gram matrix, thereby eliminating the need for re-computing $O(n^2)$ kernel function values at each evaluation of the model.

Algorithm 1 Validation. KFold validation for determining a minimum

```
1:  procedure VALIDATE_SUBSETS(𝒳, f)   ▷ Determine subsets of 𝒳 generalizable under model f
2:      for n < |𝒳| do
3:          sem := 1
4:          scores = []
5:          while 2.576 * sem < 0.01 do          ▷ enforce 3σ confidence interval of 1%
6:              Sample S ⊂ 𝒳 with |S| = n
7:              Train f on subset S
8:              Run K-fold validation on the subset (K = 5) with respect to f → s
9:              scores.append(s)
10:             sem ← Var(scores)^(1/2)/|scores|
```

Figure 5.1 shows the validated model performance of the standard HW-1 kernel circuit (introduced in Section 5.2.1).

Not only is Algorithm 1 capable of validating a QKM's performance, but it is also a tool for prototyping modified kernel circuits using simulated data. Using this technique, along with a series of trial-and-error simulated experiments, I eventually produced a prototype kernel circuit (PROTO-3) that performed significantly better on the training dataset compared to its HW-1 and ZPOW-2 predecessors (Figure 5.2).

5.2 Hardware-native kernel engineering

5.2.1 Prototyping circuits for computing quantum kernels

The structure of the quantum circuit that will produce a useful kernel using the methods of Section 4.3 is not known in advance for any given dataset. Furthermore, the structure of a quantum kernel circuit that produces good results for one dataset won't necessarily

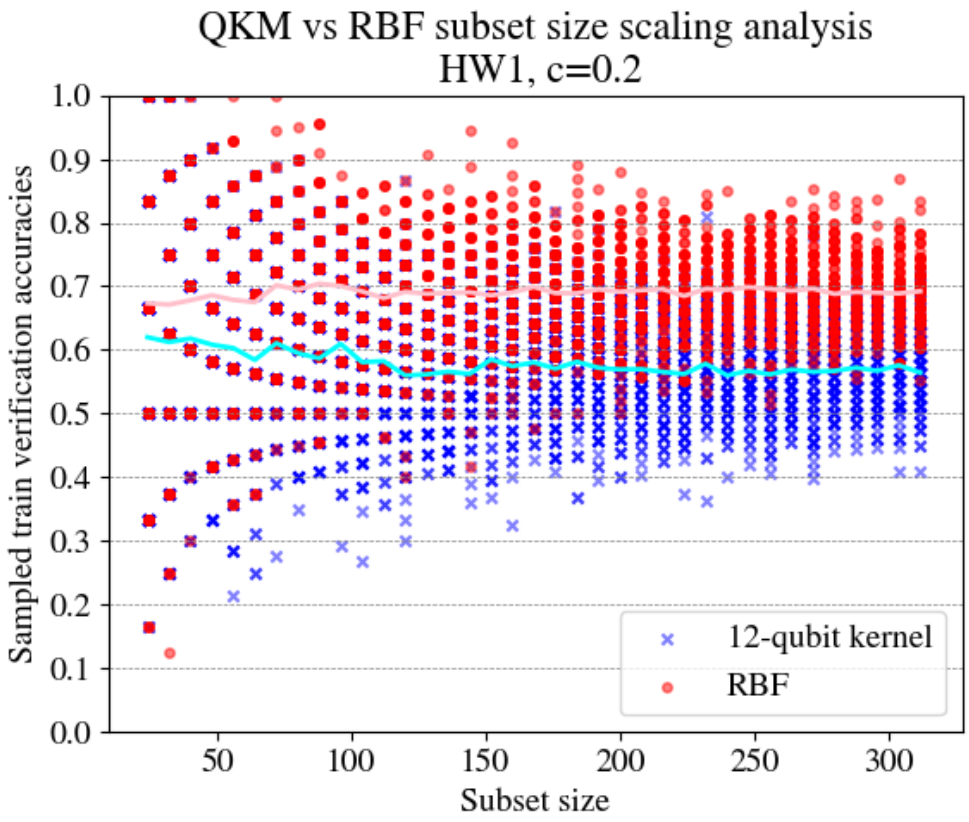


Figure 5.1: Simulated validation results for HW-1 using Algorithm 1. For each training set size on the x-axis, each point indicates a different accuracy produced by training an SVM classifier on a subset of SL PCA data that size and then evaluating its performance on a larger test set. The spread in scores associated with small train sizes is due to poor generalizability for models that have only seen a small amount of data. Even after optimization of the encoding function constants by grid search, HW-1 failed to achieve average performance anywhere near that of an RBF kernel.

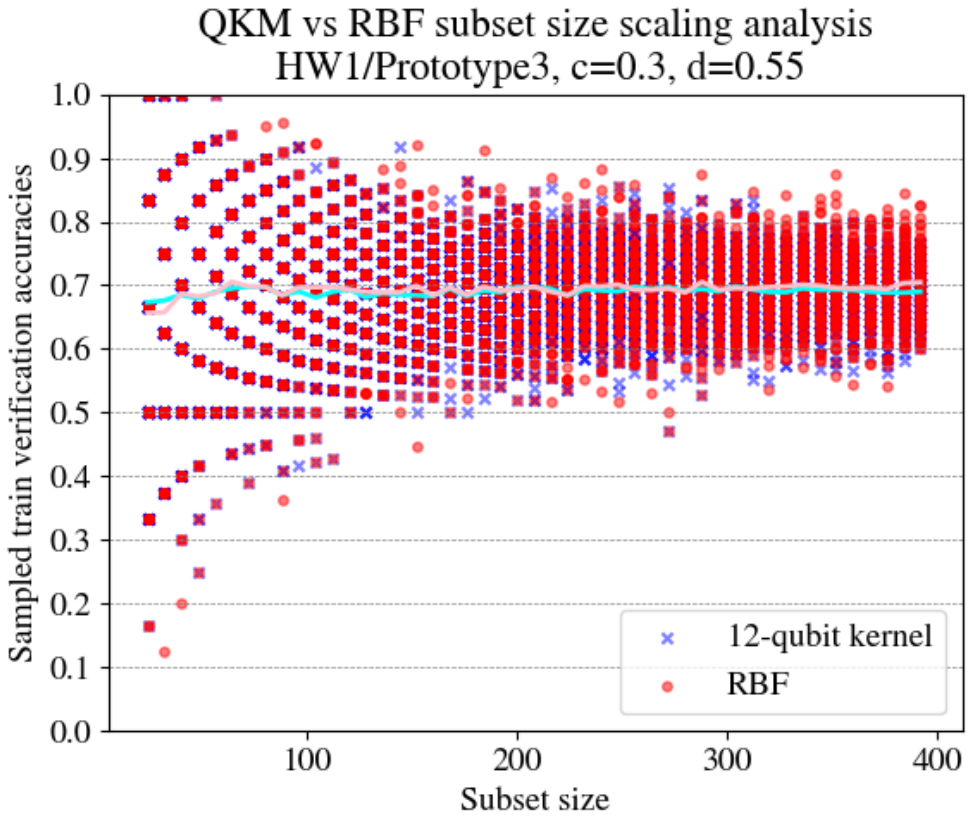


Figure 5.2: In an improvement over Figure 5.1, simulated validation results for PROTO-3 kernel circuit using Algorithm 1 show that the grid search optimized PROTO-3 circuit has performance almost indistinguishable from an out-of-the-box classical RBF kernel.

generalize to other datasets. This is also the case for classical kernel methods, which tend to be more of an art than a science. Nevertheless, its possible that high-performance kernels for a given dataset can be discovered through trial and error.

The following sections detail relevant circuit architectures and (linear) encodings used throughout this work.

ZPOW-2

This encoding circuit implements a unitary similar to the one originally proposed in [22]:

$$U_{\text{ZPOW-2}}(x) = \prod_{k=1}^{2} \left(\bigotimes_{i} H \right) \prod_{E_k(i,j)} \exp\left(\phi(i,j) Z^{(i)} Z^{(j)}\right) \tag{5.1}$$

From trial and error, it was determined localized Z-rotations did not improve accuracy on the SL dataset and were therefore dropped. Figure 5.2.1 shows a sample circuit diagram of a single layer of ZPOW-2 implemented for $n = 4$ qubits.

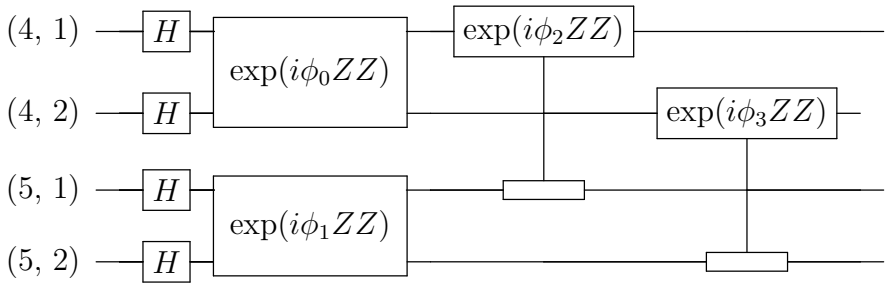


Figure 5.3: Circuit diagram implementing a single layer of ZPOW-2

HW-1

The ZPOW-2 encoding was problematic for running on actual hardware due to the depth required to implement the $\exp(i\theta ZZ)$ gate on actual hardware (see Section 5.2.2). This modified circuit replaces the $\exp(i\theta ZZ)$ gates with hardware native $\sqrt{\text{iSWAP}}$ gates followed by pairs of local rotations, resulting in a unitary of the form:

$$U_{\text{HW-1}}(x) = \prod_{k=1}^{2} \left(\bigotimes_{i} H \right) \prod_{E_k(i,j)} \sqrt{\text{ISWAP}}_{ij} R_z^{(i)}(0.2x) R_z^{(j)}(0.2x) \tag{5.2}$$

Figure 5.2.1 shows a sample circuit diagram of HW-1 implementation for $n = 4$ qubits.

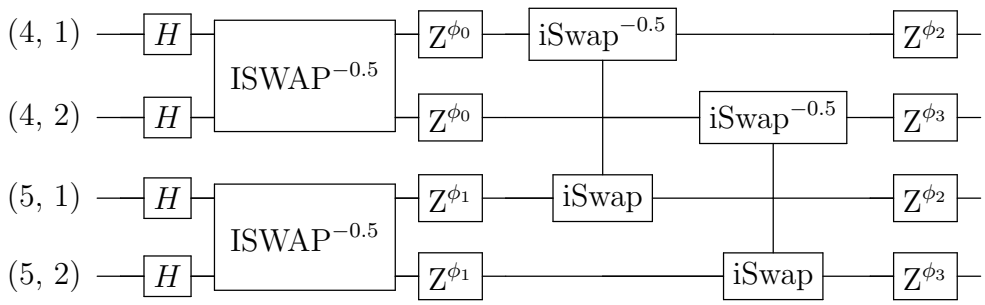


Figure 5.4: HW-1 circuit diagram

PROTO-3

Based on trial and error with the HW-1 architecture, it was eventually determined that single-qubit Z rotations *did* improve accuracy on the SL dataset. So this encoding circuit implements a unitary of the form:

$$U_{\text{PROTO-3}}(x) = \prod_{k=1}^{2} \left(\bigotimes_i H R_z^{(i)}(0.6x) \right) \prod_{E_k(i,j)} \sqrt{\text{ISWAP}_{ij}} R_z^{(i)}(0.3x) R_z^{(j)}(0.3x) \tag{5.3}$$

where the linear factors $0.6, 0.3$ were found by grid search over SVM (validated) performance (See Section 5.1.2). Figure 5.2.1 shows a sample circuit diagram of PROTO-3 implementation for $n = 4$ qubits.

5.2.2 Compilation to hardware-native gates

As mentioned in Section 5.2.1, enacting unitaries of the form $\exp(i\theta Z \otimes Z)$ on the Rainbow-23 device using the available gateset resulted in much larger depth for a given circuit. Figure 5.2.2 shows the decomposition of a single two-qubit gate of this form [1].

[1] I am grateful to D. Strain for passing this decomposition on to us.

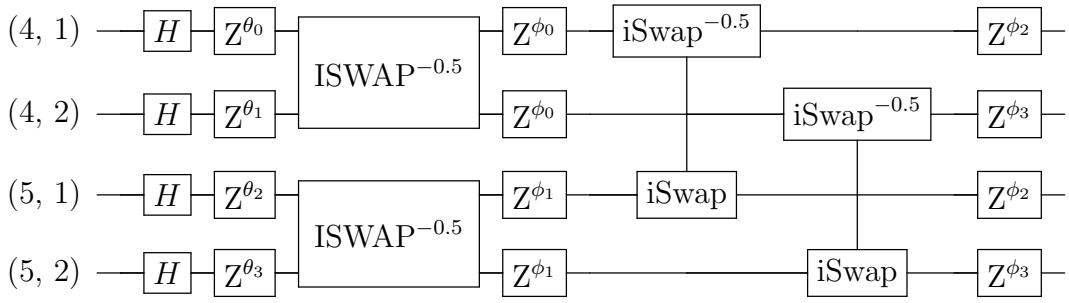


Figure 5.5: PROTO-3 circuit diagram

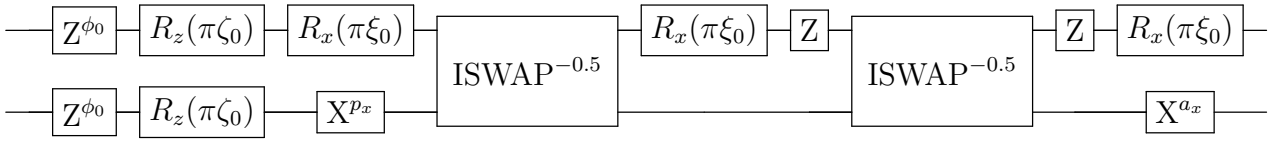


Figure 5.6: ZZPow hardware gate decomposition; each implementation of a two-qubit $\exp(i\theta Z \otimes Z)$ gate used in the circuits of [22] requires a depth-7 decomposition on Rainbow-23 (a hardware optimization on the device allows nearly instant execution of Z-gates).

5.2.3 Automated qubit map optimization

Average qubit performance is limited by design and fabrication techniques used to produce a chip. Parameters like T_1, T_2, and one- and two-qubit gate fidelities in a transmon qubit setting are all limited by extensive material defects .In addition, these parameters in superconducting qubit systems can drift on timescales as short as hours. As a result, the fidelities of gates and coherence of individual qubits are known with a large degree of uncertainty with respect to the most recent experiment determining these parameters. Therefore, the optimization procedure for preparing hardware circuits must take into account the quickly-changing knowledge of qubit performance metrics.

The qubit map algorithm constructs a qubit graph $G_q = (E, V)$ where edges represent entangling gate connectivity and nodes represent qubits. The optimization is done by traversing all simple (one to two edges per node) paths (no repeated node visits) of fixed length k (implemented according to [50, 20]), and then scoring each path according to some function of the metrics for the subset of nodes visited. Succinctly, this algorithm solves

finds the maximum:

$$\max_{f(V_k(G))} G_q \tag{5.4}$$

subject to the constraint $|V_k(G)| = k$ and $f : V \to \mathbb{R}$ is a function scoring subsets of vertices. The content of f is to implement a value system for dealing with tradeoffs between different calibration metrics.

Before applying f, the heterogeneous calibration data were normalized to the range $[0, 1]$ and inverted if they represented an error. For example, a p_1 value of 0.7% would be converted to $0.993/p_{1,max}$ while a T_1 value of 10 μs would be converted to $10/T_{1,max}$. With different datatypes homogenized, let the value of the p-th category of calibration data for the i-th qubit v_i be $c_p(v_i)$. Then a simple function f might simply be a linear function of all calibration categories:

$$f(V) = \sum_{v_i \in V} \sum_{p} c_p(v_i) \tag{5.5}$$

Figure 5.2.3 shows the results of optimization with respect to the unweighted f. In particular, the path included an undesirable qubit $q10$ in order to reach the highest-performing qubit $q11$. In practice, this might lead to poor performance of the algorithm due to including the "weak link" $q10$, and introduces the need to assign a penalty to bad calibration values. This modifies the form of f to be more general:

$$f(V) = \sum_{v_i \in V} \sum_{p} g_p(c_p(v_i)) \tag{5.6}$$

where g_p is a scoring function applied to the p-th normalized, inverted calibration metric. An example of the results of weighting c_{T1} to penalize low T_1 values is shown in Figure 5.2.3, alongside the logarithmic penalty function for normalized T_1 values in Figure 5.2.3. This modified graph now avoids the exceptionally bad $q10$, demonstrating the flexibility of using weighted scoring function for qubit selection.

Automated qubit map optimization as described here was implemented in all later iterations of experiments after February 17.

5.3 Data postprocessing

This section uses the notation $\underline{0}^n$ to represent the all-zeros bitstring over n qubits. In general, underlined $\underline{0}$'s and $\underline{1}$'s will be used to represent individual bits of $\{0, 1\}$, and underlined bitstrings of length k will represent vectors in $\{0, 1\}^k$.

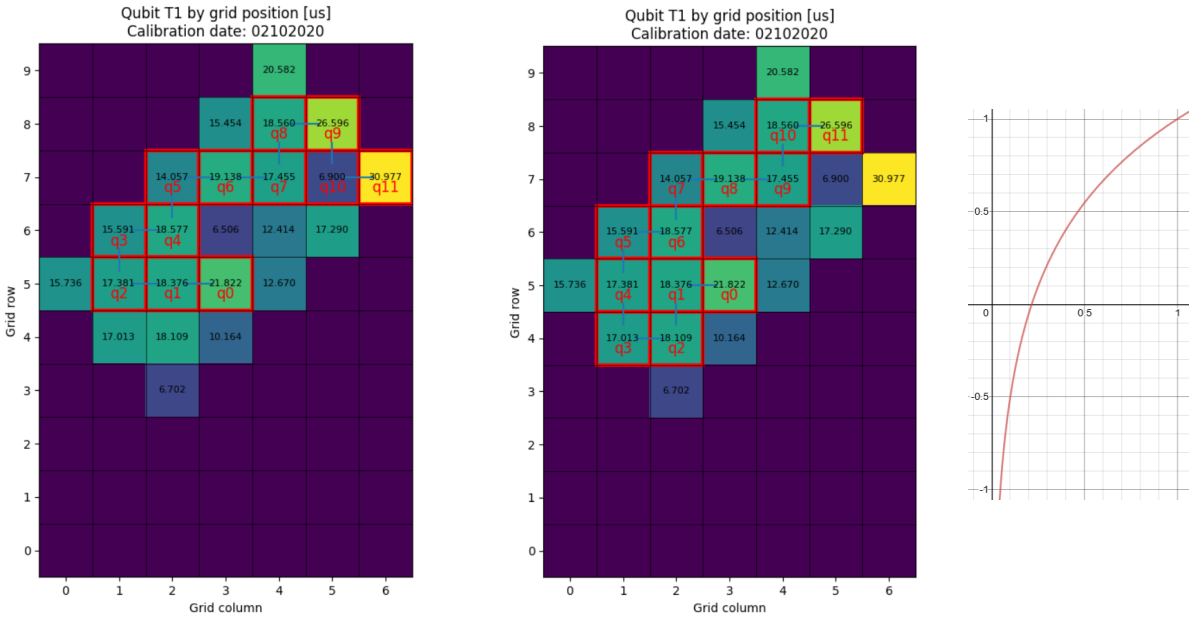


Figure 5.7: (left) The results of automatically optimizing 12-qubit "snake" grid layout (red boxes) with respect to unweighted calibration metrics, overlaid on T_1 values serving as input to the optimizer. (center) The same grid structure generated by using a logarithmic penalty function applied to T_1 values. (right) An plot of the penalty function $y = 1.5\ln(x) + 1$ applied to T_1 values to achieve the results in (center).

As introduced in Section 4.3, the kernel matrix element $k(x, x')$ is determined from the probability $P(0^n)$. Due to the measurement error described in Section 2.3.2, increasing the number of qubits results in an exponentially higher rate of transition out of the $\underline{0}^n$ state $N_{\underline{0}^n} \prod_i^n p_0^i$, accompanied by a higher transition rate into the $\underline{0}^n$ state from all states some Hamming distance h away from $\underline{0}^n$.

5.3.1 Bitflip correction by system of linear equations

Let N_i be the prior frequency of bitstring 'i' for an experiment with no bitflip error, i.e. $p_0^q = p_1^q = 0$ for all qubits $\{q\}$. Let M_i be the observed frequency of bitstring 'i' and define an offset x_i so that $N_i = M_i + x_i$. Now define a response matrix $(P)_{jk}$ that contains as its elements the total probability for transition from bitstring 'k' to bitstring 'j' (for example, Figure 5.8 shows an example response matrix for $n = 6$ qubits). P is not symmetric in general, since typically $p_0 < p_1$.

The goal is to solve for x_i given P and M_i for all bitstrings b_i as a system of linear equations. This follows from the statement that the observed frequency of bitstring k is the "true" frequency of k reduced by transitions $k \to j$ and increased by transitions $j \to k$ from all prior populations $j \neq k$:

$$M_k = N_k(1 - \sum_{j \neq k} P_{jk}) + \sum_{j \neq k} N_j P_{kj} \tag{5.7}$$

$$= (M_k + x_k)P_{kk} + \sum_{j \neq k} (M_j + x_j)P_{kj} \tag{5.8}$$

This yields a linear equation of the form $A\vec{x} = \vec{b}$ where $A = P^T$ and $B = (1 - P^T)\vec{M}$. In practice, drift or inaccuracies in the recorded bit flip probabilities or statistical uncertainty introduced by finite sample sizes can lead to inconsistencies with the observed bitstring frequencies M (an example is if M_k is very large but P_{kk} is vanishing). This can produce skewed or negative predictions for frequencies of bitstrings in the prior distribution that must be corrected by undesirable rounding and up-/down-sampling.

5.3.2 Bitflip correction by Bayesian iterative unfolding

The method of Section 5.3.1 is unstable due to the possibility that matrix inversion yields negative-valued entries (which will possibly result in negative calculated prior frequencies).

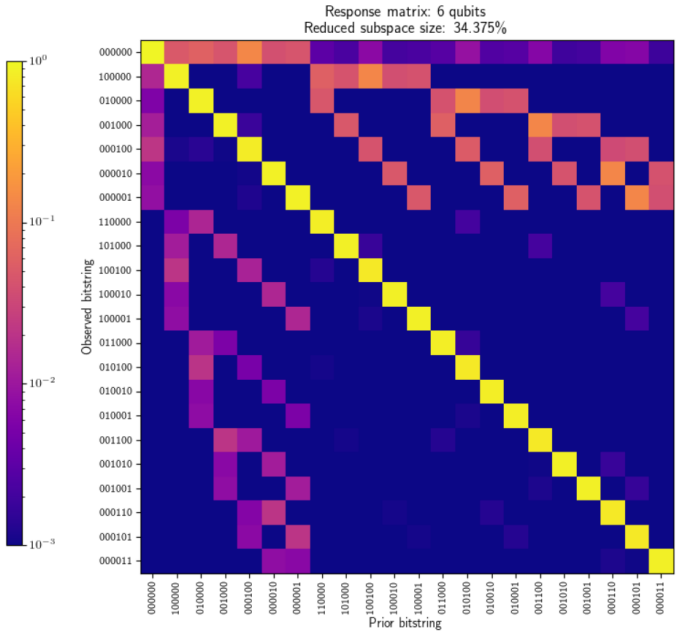


Figure 5.8: The response matrix characterizing total probabilities for transitions between length-6 bitstrings, computed from 3/31/2020 calibration data. The logarithm of probability for transition from a bitstring C_j on the vertical axis to a bitstring E_i on the horizontal axis is displayed as the color of the matrix element. Nonzero off-diagonal elements are the source of bitflip error, and the asymmetry in the response matrix is caused by the asymmetric bitflip probabilities for single bits. The set of bitstrings of weight 2 or less shown represents 34% of all length-6 bitstrings but accounts for almost all transitions into/out of the all-zeros bitstring.

Table 5.1: Likelihoods for single bit bitflips based on p_0 and p_1 values. These single-bit bitflip likelihoods are combined to produce full bitstring transition likelihoods according to Equation 5.9.

b_C	b_E	$P(b_E\|b_C)$
0	0	$1-p_0$
0	1	p_0
1	0	p_1
1	1	$1-p_1$

For a more stable correction scheme, I implemented an iterative unfolding method as first introduced in [14]. Using their notation, define:

- C_i: An unobserved ("true") bitstring i labeled "cause"
- E_i: An observed bitstring i labeled "effect"
- The likelihood that bit i is observed from an underlying cause j, $P(E_i|C_j)$
- An unknown "prior" $P(C_i)$ describing the "true" frequency of bitstring i

The likelihood $P(E_i|C_j)$ is identical to the element P_{ij} from the response matrix introduced in Section 5.3.1. The likelihood of transition for a single cause bit b_C into an effect bit b_E is given by the truth Table 5.1. The transition likelihood for a length-n bitstrings is then just a product of the transition likelihoods for the individual bits that make up the bitstring:

$$P(E_i|C_j) = \prod_{k=1}^{n} P((E_i)_k|(C_j)_k) \tag{5.9}$$

Where $(E_i)_k, (C_j)_k$ denote the k-th bit of the effect and cause bitstring respectively. An intermediate requirement for performing the unfolding is the probability $P(C_i|E_j)$ which is connected to the above elements via Bayes' theorem[2]:

$$P(C_i|E_j) = \frac{P(E_j|C_i)P(C_i)}{\sum_{\ell=1}^{n_C} P(E_j|C_\ell)P(C_\ell)} \tag{5.10}$$

[2]It's tempting to try to solve for $P(C_i|E_j)$ directly using Table 5.1, thereby avoiding the need for Equation 5.9. This would be a mistake, since the bitflip probabilities $p_0 = P(\text{"effect = 1"}|\text{"cause = 0"})$ and $p_1 = P(\text{"effect = 0"}|\text{"cause = 1"})$ are themselves *likelihoods* conditioned on causes and arising from physical processes.

The problem statement for Bayesian iterative unfolding is then as follows: **Given** knowledge of how many bitstrings j were observed, $P(E_j)$, and the known transition probability $P(E_i|C_j)$ from an unseen *cause* bitstring C_j to an observed *effect* bitstring E_i, **find** the unknown prior distribution of unseen bitstrings $P(C_i)$. The method of [14] was implemented using the following iterative procedure, using the PYUNFOLD library [10]:

1. Compute the response matrix $P_{ij} = P(E_j|C_i)$ from calibration data for bitflip probabilities $p0$, $p1$ and Equation 5.9
2. Guess an initial prior distribution bitstrings, $P^{(0)}(C_i)$
3. Compute an initial estimator for the frequency of C_i using observed frequencies for E_j along with the computed $P(E_j|C_i)$ from step (1) and Equation 5.10:

$$\hat{n}^{(0)}(C_i) = \sum_{j=1}^{n_E} P(C_i|E_j) n(E_j) \tag{5.11}$$

4. Compute an estimator for $P(C_i)$: $\hat{P}(C_i) = \hat{n}^{(0)}(C_i) / \sum_i \hat{n}^{(0)}(C_i)$
5. Compare $\hat{P}(C_i)$ to $P^{(0)}(C_i)$:
   - if $\hat{P}(C_i) \approx P^{(0)}(C_i)$, terminate.
   - otherwise, set $P^{(0)}(C_i) = \hat{P}(C_i)$ and repeat steps 3-5

Hamming truncation

An immediate issue with the technique of Section 5.3.2 is that it requires a response matrix defined over all of the relevant bitstrings, which scales exponentially with the number of qubits in the experiment. However, since the goal of the experiment is to determine the probability of readout for $\underline{0}^n$, and observing that $P_{ij} << 1$ causes the probability of transitions between $0^{N_{\text{qubits}}}$ and a bitstring Hamming distance k away is suppressed exponentially in k, so bitstrings with a large Hamming distance from the all-zeros string can be safely excluded during the correction.

Approximating the complexity of bitflip correction via iterative unfolding as the same as that of matrix inversion (Section 5.3.1), then the naive matrix inversion complexity of $\mathcal{O}(n^3)$ for an $n \times n$ matrix results in exponential complexity of $\mathcal{O}(2^{3d})$ for iterative unfolding with respect to d qubits. However, if correction is restricted to only bitstrings of weight

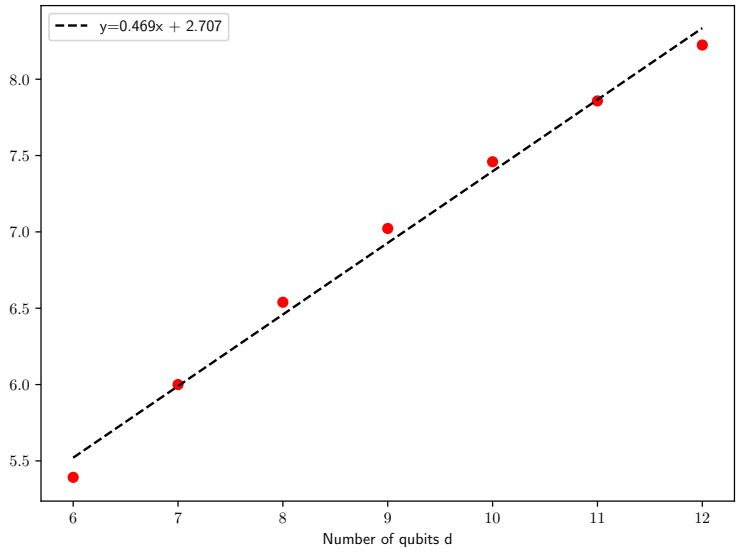


Figure 5.9: The size of a bitstring subspace restricted to bitstrings of weight four or less is given by $\log\left(\sum_{k=0}^{4} {}_{d}C_{k}\right)$ as a function of number of qubits d. While still exponential in d, the dimensionality of the space of bitstrings of weight ≤ 4 scales like $2^{0.47d}$ for $d < 13$, a factor of roughly $2^{d/2}$ less than the dimensionality of the full space of bitstrings.

less than or equal to some constant, the exponent of $3d$ can be linearly reduced. Figure 5.9 demonstrates that for the case of truncating to bitstrings of weight ≤ 4 the exponential argument for the complexity of bitflip correction can be reduced by a factor of about $\frac{1}{2}$. In other words, this results in $2^{d/2}$ reduction in overhead for correcting bitflip errors in practice for $d \leq 12$.

Given a set of conditional likelihoods $P(E_i|C_j)$, a desired cause C_x to compute, and the set of bitstrings of weight equal to k, $\{v_k\} \equiv \{v \in \{0,1\}^n : |v| = k\}$, let $\epsilon_{x,k}$ be the net error introduced into the likelihoods of the form $P(E_x|C_j)$ due to excluding all bitstrings of weight greater than k:

$$\epsilon_{x,k} \equiv 1 - \sum_{C_j \in \{v_k\}} P(E_x|C_j) \tag{5.12}$$

The sum in Equation 5.12 is simply the sum of the column corresponding to E_x in the response matrix that includes only transitions into E_x from bitstrings of weight k or less (recall that $\sum_j P(E|C_j) = 1$ when no entries are excluded; as k increases Equation 5.12 will asymptotically approach 1). Similarly, let $\epsilon_{k,x}$ be the error introduced to likelihoods

of the form $P(E_j|C_x)$ due to Hamming truncation:

$$\epsilon_{k,x} \equiv 1 - \frac{\sum_{E_i \in \{v_k\}} P(E_i|C_x)}{\sum_{j=1}^{n_E} P(E_j|C_x)} \tag{5.13}$$

The sum in Equation 5.13 computes the normalized sum of the row corresponding C_x for a Hamming-truncated response matrix; since $\sum_i P(E_i|C) \neq 1$ in general, this sum is normalized according to sum of the same row in the non-truncated version of the response matrix. Equations 5.12-5.13 can be computationally expensive to compute for many qubits since the size of the response matrix scales like 2^n. A very rough approximation of these errors can be computed directly from p_0, p_1 bitflip probabilities by using the following heuristics:

$$\epsilon_{x,k} \approx 1 - \sum_{\ell=k+1}^{n/2} \binom{n}{\ell} \overline{p_0}^{\ell} \tag{5.14}$$

$$\epsilon_{k,x} \approx 1 - \sum_{\ell=k+1}^{n/2} \binom{n}{\ell} \overline{p_1}^{\ell} \tag{5.15}$$

where $\overline{p_0}$ is the mean probability for a bitflip from "0" to "1". Conceptually, the sums in Equations 5.14-5.15 treat transition into/out of any bitstring of weight ℓ as equally likely and accumulate the error introduced by ignoring bitstrings of weight greater than the set truncation point, and the termination of the sum at $n/2$ coincides with the maximum of n-choose-ℓ.

Figure 5.10 provides an example illustration of appropriate cutoffs for computing the all-zeros bitstring using the exact errors of Equations 5.12-5.13 as a function of k for different numbers of qubits n for a specific set of bitflip probabilities, as well as the heuristic Equations 5.14-5.15. The approximation for $\epsilon_{k,x}$ tends to diverge from the true value for large numbers of qubits, but captures the asymptotic behavior necessary to determine the Hamming distance at which to truncate bitstring transitions in a truncated response matrix.

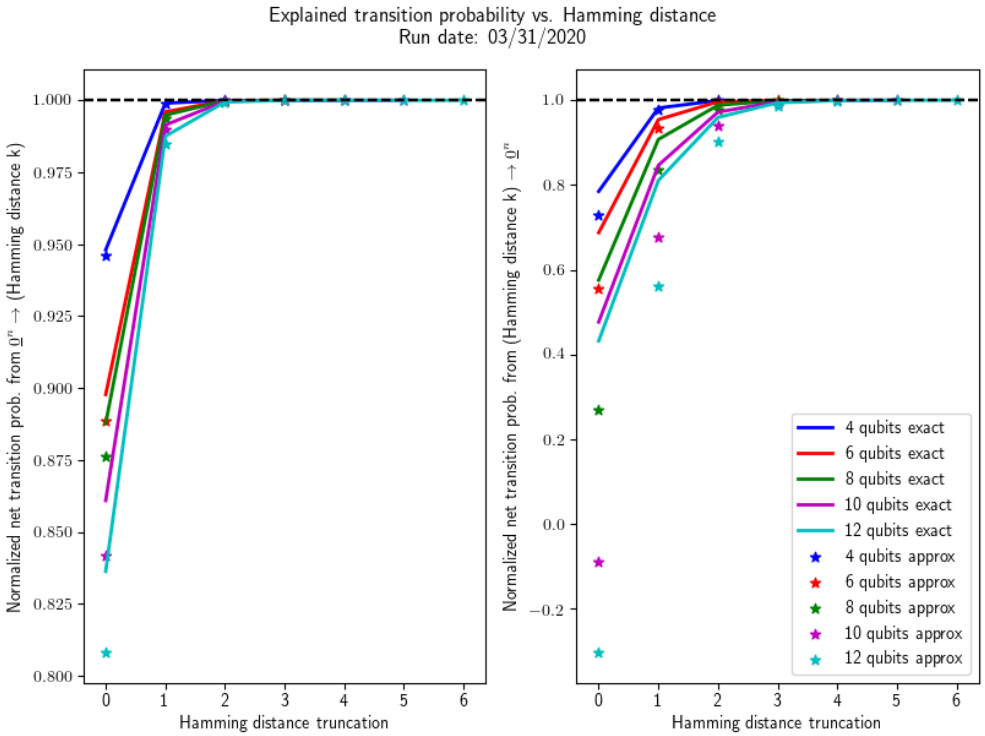


Figure 5.10: (left) $1 - \epsilon_{x,k}$ and (right) $1 - \epsilon_{k,x}$

Table 5.2: Abbreviations for distributions to be compared in the following analyses.

Label	Description
SIM	Noiseless simulation
NSY	Full noisy simulation
HW	Hardware
NSY~BF	Noisy simulation that excludes bitflip errors
SIM+BF	Bitflip-only noise simulation
CORR[·]	Bitflip-corrected version of argument (with Hamming truncation as indicated)

5.4 Assessing bitflip correction performance and uncertainty

By providing the full noise model introduced in Section 2.3.5 with a complete set of calibration data for all relevant parameters (see Appendix A), one can characterize the performance of the model by bitstring distributions to the hardware outcomes. On a very low level this can be accomplished by comparing the full histograms of bitstrings observed in the noiseless simulation, noisy simulation, and hardware runs. To capture this comparison so that broader trends in performance can be observed, differences in distributions of bitstrings sampled from various sources will be summarized using the Kullback-Leibler divergence (also referred to as KL-divergence,D_{KL}, or relative entropy) of the distributions:

$$D_{KL}(P||Q) \equiv -\sum_{x} P(x) \log Q(x) - S(P) \tag{5.16}$$

where S is represents the entropy and P, Q are discrete distributions. Since training an SVM using the quantum kernel method requires access to a gram matrix of sufficient size to return validated accuracy scores (see Section 5.1.2), then computing D_{KL} over the full gram matrix gives a reasonable estimate of how well hardware results conform to simulated expectations.

Controlled analysis of the performance of the bitflip correction scheme can be accomplished by comparing different variations of simulated noise models and hardware outcomes with and without the correction applied. Table 5.4 introduces identifiers that will be applied during the analysis.

The following sections explore the performance of the bitflip correction algorithm by comparing various distributions in Table 5.4. Additionally, there is a very real risk of the asymmetric measurement error producing an excess of all-zeros bitstrings compared to the

true rate, which might suggest that the decoherence in the circuit is not as bad as it really is. To control for this, I provide results of experiments in which the input state of the kernel circuit was prepared to $|\underline{1}^n\rangle$. This then maps the value of $k(x, x')$ onto $P(\underline{1}^n)$, which will be suppressed due to asymmetric measurement error to the same degree that $P(\underline{0}^n)$ will be artificially increased.

As a reminder, the notation $\underline{0}^n$ and $\underline{1}^n$ used to represent the all-zeros and all-ones bitstrings over n qubits. The histograms in the following subsections all have the following properties:

1. Histograms are restricted to the sorted top 12 frequencies with respect to the union of the compared distribution (though the $\underline{0}^{12}$ bitstring always appears first).

2. **Bitflip correction skew** - When analyzing these types of histograms, the following t is very important to remember that **Hamming truncation skews bitflip correction results according to the Hamming distance from the un-modified value**. That is, after the methods of the previous section have been applied to a distribution, the accuracy of the resulting $P(\underline{0}^n)$ (or $P(\underline{1}^n)$ in the case of the inverted input state) will be much higher than the accuracy of a Hamming distance one bitstring (e.g. $P(\underline{0100})$ for $n = 4$) since the Hamming truncation leaves an asymmetric bitstring transition space with respect to such a bitstring.

5.4.1 CORR[HW] vs NSY~BF

Comparing CORR[HW] to NSY~BF determines the performance of the simulation noise model. Given a reliable algorithm for bitflip correction, this quantity describes how closely the dynamics *within* the circuit are being modeled with respect to the true hardware dynamics.

Figures 5.11-5.12 present two examples of CORR[HW] distributions compared to their NSY~BF counterparts for 12 qubits. Figure 5.11 corresponds to the computation of a diagonal kernel element (for which $P(\underline{0}^n) = 1$ in noiseless simulation) with X gates inserted on all qubits immediately before readout (to bias against the tendency of asymmetric bitflip probabilities to cause "pileup" in the $\underline{0}^n$ bitstring); Figure 5.12 shows another 12-qubit event for an offidiagonal kernel element with no X gate modification.

The qualitative comparison shown here generalizes to most 12-qubit results: There is only sporadic agreement between CORR[HW] and NSY~BF for 12 qubits, which indicates that a large part of the hardware results are due to noise processes that cannot

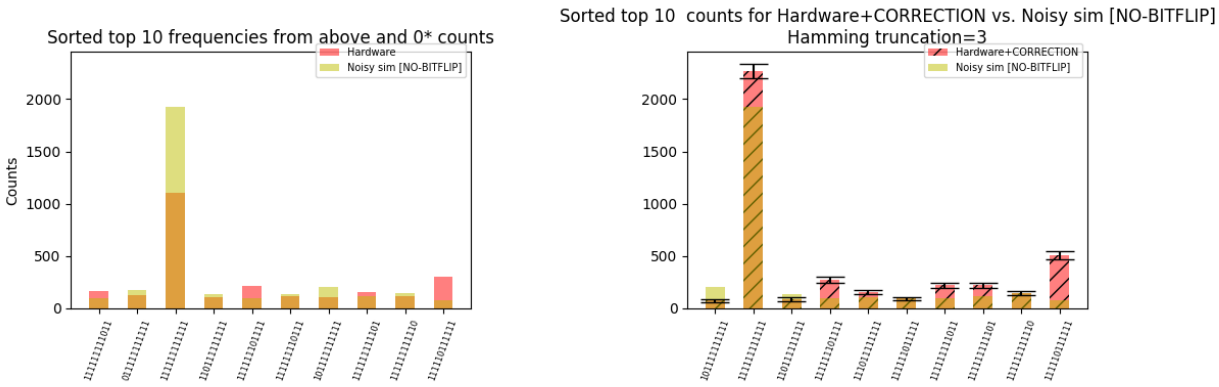


Figure 5.11: Sample distribution for computation of a HW-1 diagonal element on 12 qubits, for (left) HW vs NSY~BF and (right) CORR[HW] vs NSY~BF. Recall that due to the Hamming truncation procedure, only the $\underline{1}^{12}$ bitstring population on the right histogram is valid. The bitflip correction algorithm is generally successful at recovering populations for the $\underline{0}^{12}$ / $\underline{1}^{12}$ bitstrings for diagonal elements, likely due to the large populations of these bitstrings to begin with.

be accounted for using the present noise model, and motivates against the use of 12 qubits for full runs used to train an SVM.

5.4.2 CORR[HW] vs SIM

Comparing CORR[HW] to SIM determines the verifiability of the results received from hardware with respect to the broader QKM model. In practice, the anticipated difference in performance of an SVM trained using a Gram matrix computed from hardware versus noiseless simulation will depend on the un-correctable statistics for the $0 \ldots 0$ bitstrings in each case.

Figure 5.13 shows comparisons between (subsets of) the distributions observed for HW, CORR[HW], and SIM respectively for a few example distributions taken from a subset of $k(x_i, x_j)$ computed using PROTO-3 for $n = 8$ qubits. There is a general improvement in the agreement between hardware outcomes and noiseless simulation after the correction has been applied.

By comparing these outcomes like Figure 5.13 over the full set of $k(x_i, x_j)$ computed, a distribution of KL-divergence values can be assembled showing a general trend of improve-

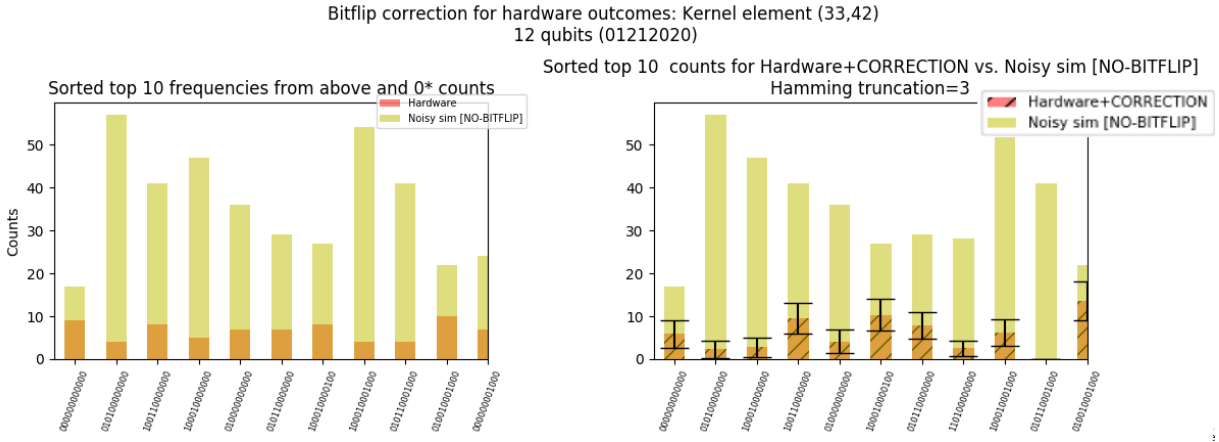


Figure 5.12: Sample distribution for computation of a HW-1 off-diagonal element on 12 qubits, for (left) HW vs NSY~BF and (right) CORR[HW] vs NSY~BF. A number of different off-diagonal circuits exhibited this behavior

ment in the agreement between CORR[HW] and SIM compared to the agreement between HW and SIM. Figure 5.14 demonstrates this overall trend in improvement for three separate experiments: an unmodified PROTO-3 circuit, a modified version with $n/2$ X-gates over half of the qubits inserted at the beginning of the circuit, and a modified version with n X-gates over all of the qubits inserted (again, the motivation for these modifications is to control for any possible bias introduced into sampled outcomes due to the asymmetric nature of the readout error, though this has a much smaller effect on comparisons of full distributions than comparisons of $P(\underline{0}^n)$).

Plotting mean D_{KL} values taken from a set of histograms like Figure 5.14 for $n = 4, 6, 8, 10$ qubits (10 being the maximum number of qubits for which iterative unfolding can be performed over *full* distributions of outcomes using the available implementation) yields the scaling analysis in Figure 5.15. While D_{KL} cannot provide an absolute measure for how successful a model employing QKM will be, the scaling in Figure 5.15 provides two insights:

1. For all numbers of qubits and independently of artificial bitflips introduced to the circuit, bitflip correction improves the performance of hardware outcomes.

2. There is an accelerating trend in the disagreement of bot HW and CORR[HW] outcomes versus simulation with respect to the number of qubits.

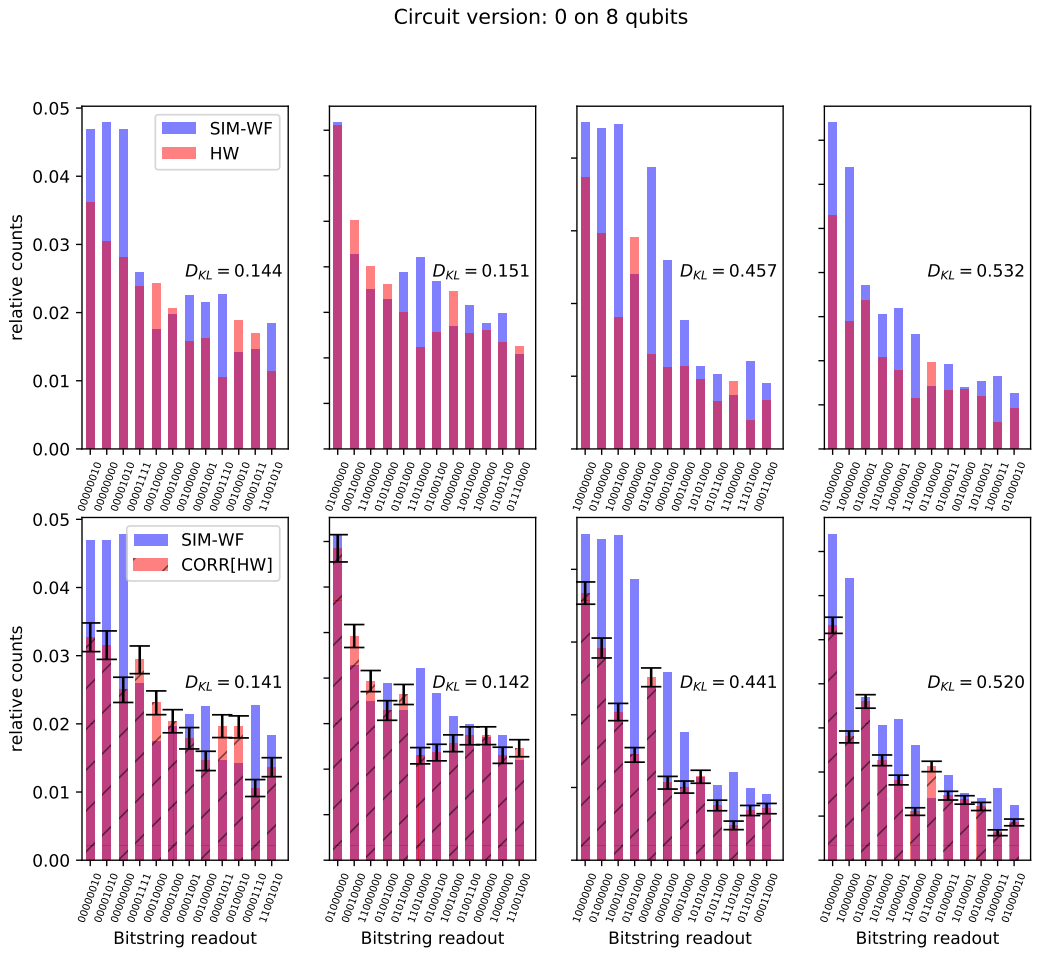


Figure 5.13: Sample distributions for HW vs. SIM on an 8 qubit PROTO-3 architecture before and after the full distributions were bitflip corrected. This visualization only includes the most frequent (with respect to the union of the distributions) outcomes of each bitstring histogram. These examples also highlight shortcomings of D_{KL} as a comparison metric: Even a relatively small KL-divergence can correspond to a large error in the computed probability of the all-zeros bitstring (as is the case in the bottom left pane, third histogram bin).

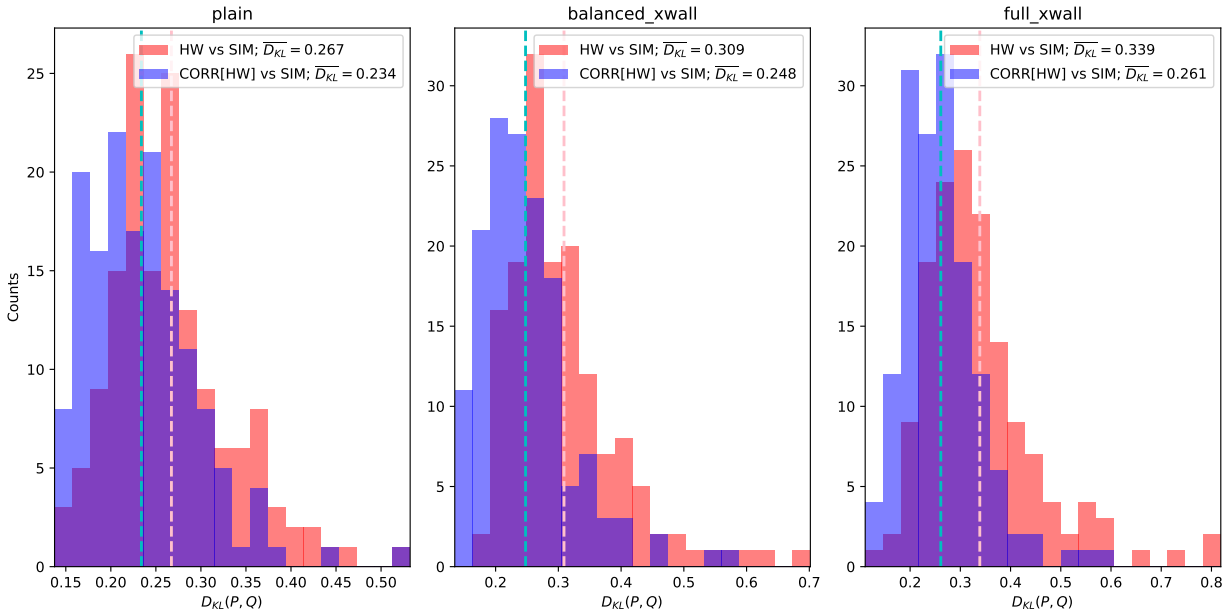


Figure 5.14: Comparing KL-divergence over larger sets of PROTO-3 circuits that have (left) no modification (middle) 4 artificial bitflips introduced and (right) 8 artificial bitflips introduced demonstrates CORR[HW] has better agreement with SIM than HW in general.

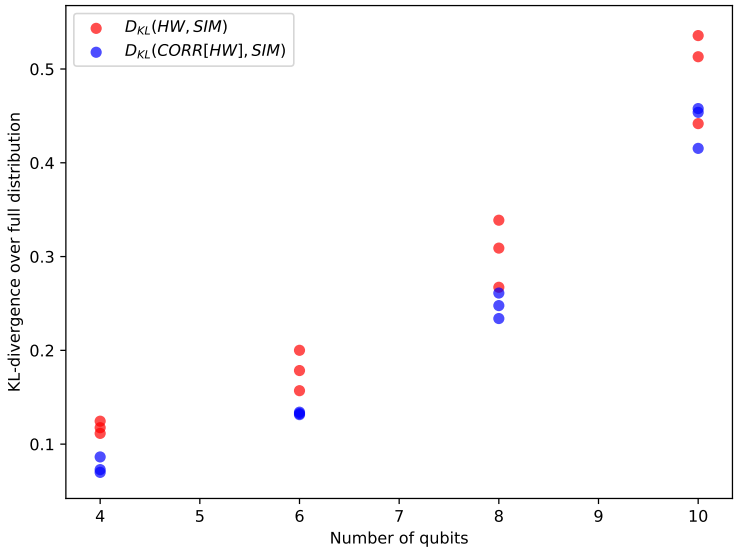


Figure 5.15: The mean KL-divergence from each artificial bitflip scenario on PROTO-3 circuits trends steadily upwards with respect to the number of qubits when comparing either HW to SIM or CORR[HW] to SIM. Assuming that the Bayesian iterative unfolding corrects a large fraction of the bitflip error, this is evidence that the remainder of hardware noise sources contribute to degradation in performance that scales super-linearly with the number of qubits. The stopping point of 10 qubits represents the largest system for which un-truncated bitflip correction can be performed using the available software.

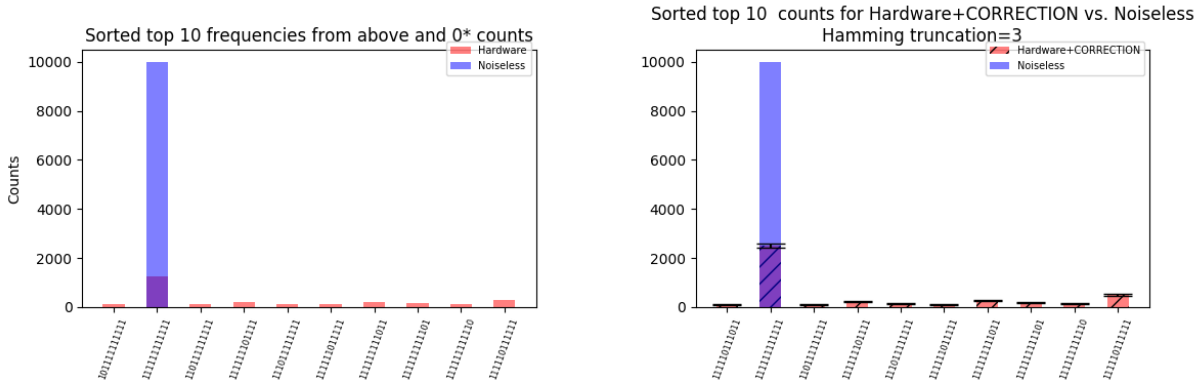


Figure 5.16: The outcome in this sample comparison of CORR[HW] vs SIM generalizes to most other 12 qubit results: Only a small part of the deficit in $P(\underline{0}^{12})$ bitstring is explained by readout error, and attempting to use even the bitflip corrected results in an SVM will result in a model performance that is completely uncorrelated with the performance of the quantum hardware. Distributions represent a diagonal element computed using 12 qubit HW-1 architecture modified with a full set of X gates immediately before readout.

Attempting to extend this comparison to 12 qubits (for which correction of the full distribution of bitstrings is no longer possible), Figure 5.16 compares sample values for $P(\underline{1}^n)$ computed using truncation on bitstrings Hamming distance > 3 away from $\underline{1}^{12}$ (the target bitstring is all one's due to the artificial bitflip circuit modification). The main observation gathered from this example and others like it is that for results sampled from diagonal element circuits the bitflip errors apparently account for only a small fraction of the observed deficit in $P(\underline{0}^{12})$ and $P(\underline{1}^{12})$. This suggests a restriction on scaling up to 12 qubits imposed by the hardware, which combined with results presented in Section 5.6 suggests that 12 qubits presents an insurmountable barrier to computing HW-1 circuit outcomes on the SL dataset.

5.4.3 CORR[SIM+BF](0) vs. SIM(0)

This comparison is meant to isolate the performance of the bitflip correction scheme by controlling for all sources of error except bitflips in a simulated setting. The inequality

$$[\text{CORR[SIM+BF]}(0) - \text{SIM}(0)] < \epsilon$$

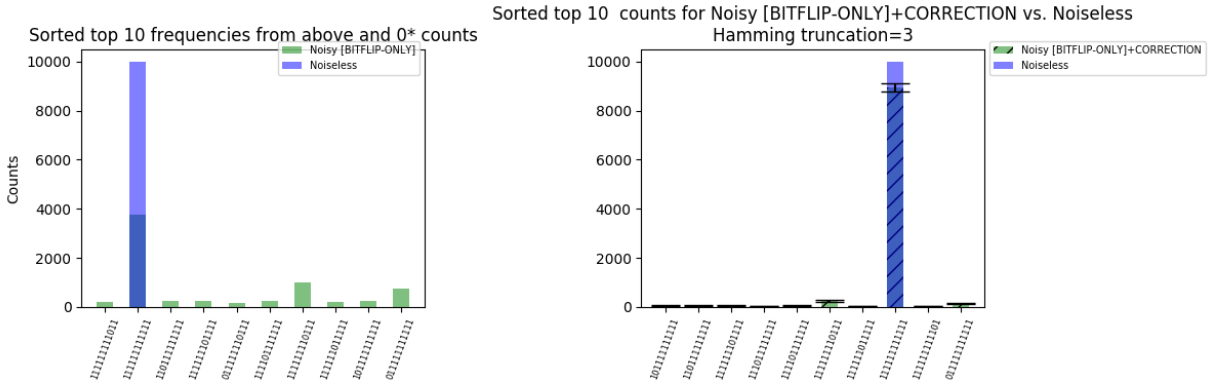


Figure 5.17: For a diagonal element (plus artificial bitflips before readout) Hamming-truncated bitflip correction *almost* recovers the SIM frequency of $\underline{1}^{12}$, but fails to do so within the combined uncertainty of Hamming truncation and p_0, p_1 statistical error.

will be satisfied if the performance of the bitflip correction scheme is accurate to within its propagated uncertainty bound ϵ, which must account for both statistical uncertainty in p_0 and p_1 as well as error introduced by Hamming truncation (as provided in Figure 5.10, for example).

From Figure 5.17 and other comparisons like it, its apparent that while the bitflip correction method developed in this chapter seems to improve the hardware outcomes, the propagated uncertainties fail to provide meaningful error bars on the result of the bitflip correction. Tuning the error propagation methods described in [14, 10] will be an important subject of future work.

5.4.4 CORR[NSY](0) vs NSY~BF(0)

Similarly to the previous section, comparing CORR[NSY](0) to NSY~BF(0) provides an understanding of how well the bitflip correction performs in a simulated setting where more general (but still well controlled) noise is included. The inequality

$$|\text{CORR[NSY]}(0) - \text{NSY}\sim\text{BF}(0)| < \epsilon$$

will be satisfied if the performance of the bitflip correction scheme is accurate to within its stated uncertainty bounds ϵ.

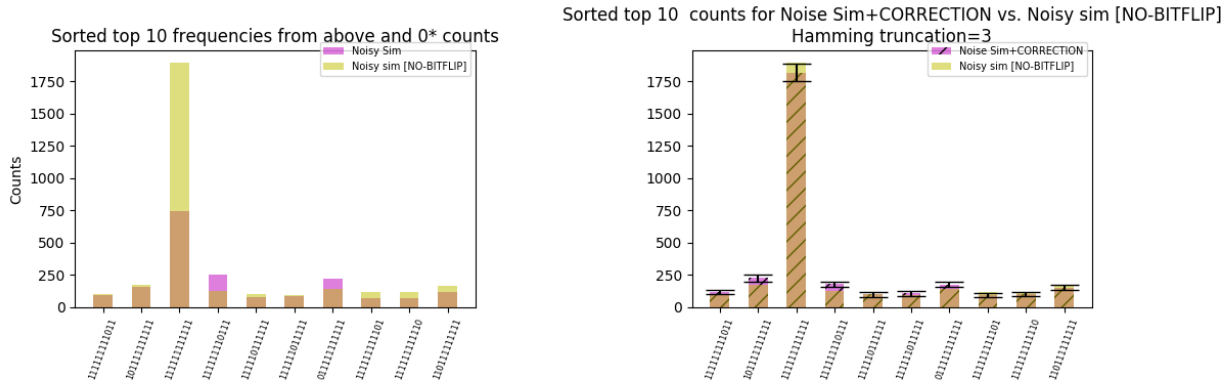


Figure 5.18: This example comparison between CORR[NSY](0) and NSY~BF(0) for a diagonal element (sampled from 12 qubit HW-1 circuit architecture) shows that even in the presence of non-bitflip noise, the bitflip correction tends to perform decently well.

In the right panels of Figures 5.18-5.19 the corrected value for $P(\underline{1}^{12})$ (or $P(\underline{0}^{12})$) computed from a full noisy simulation agrees well with the corresponding value in the noisy simulation that had no bitflips to begin with. This furthers the hypothesis that the bitflip correction scheme is effective if provided a distribution for which the primary error mechanism was bitflip error. In the examples provided, the propagated error bounds for the bitflip correction scheme also capture the degree of disagreement between CORR[NSY](0) and NSY~BF(0), though a more thorough study over a larger dataset is necessary before generalizing this behavior to the wider dataset.

5.5 Characterizing quantum SVM classifier performance on quantum hardware

I used a 12 qubit implementation of the PROTO-3 architecture to compute a 200×200 Gram matrix corresponding to the training set of the SL dataset. Figure 5.20 shows the performance of an SVM trained on the bitflip-corrected hardware outcomes for $P(\underline{0}^{12}) = k(x_i, x_j)$ using the validation scheme described in Algorithm 1 with 5-fold validation. Like its simulated counterpart in Figure 5.2, the QKM SVM achieves comparable accuracy to an RBF kernel.

The accuracy outcomes in Figure 5.20 demonstrate limited success of the QKM clas-

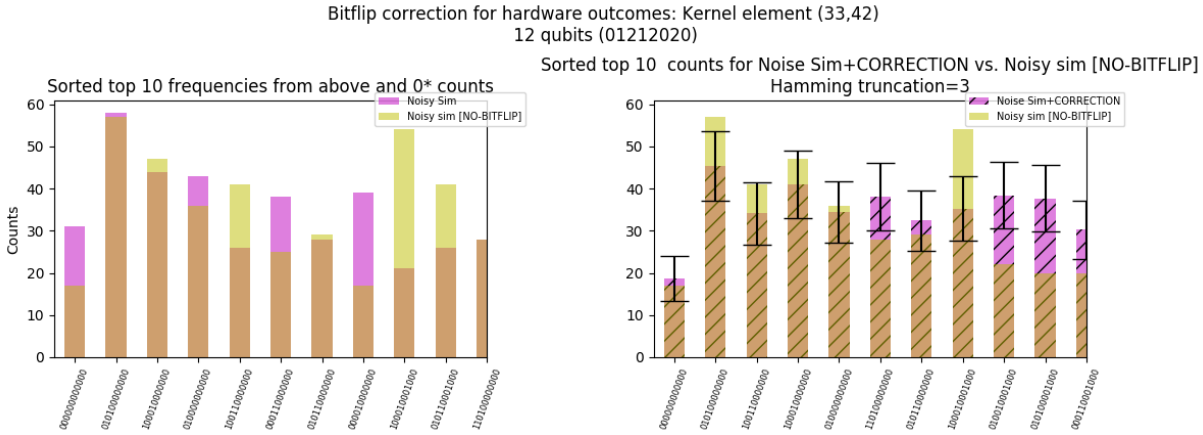


Figure 5.19: Another example comparison between CORR[NSY](0) and NSY~BF(0) for an off-diagonal element (sampled from 12 qubit HW-1 circuit architecture) showing that underlying noise in a system doesn't necessarily corrupt the bitflip correction algorithm.

sifier. This is due to the imbalanced nature of the SL dataset used in this work, wherein the population of events with the label $y = 1$ (corresponding to "not lensed") represented roughly 70% of the full population of the dataset. This leads to a situation in which even a very bad classifier that always guesses a class label of $y = 1$ can perform with 70% accuracy on representative subsets of the SL data.

In addition to the data balancing problem, the error in the computed probabilities $P(\underline{0}^{12}) = k(x_i, x_j)$ for the 12 qubit circuit is heavily skewed towards events with limited statistics for the all-zeros bitstring. Figure 5.21 summarizes this behavior by demonstrating that roughly 50% of the cumulative error across all computed $P(\underline{0}^{1}2)$ values is associated with sampled distributions with less than 10 occurrences of the all-zeros bitstring.

The final section of this thesis is devoted to explaining this behavior and provides some evidence that the observed errors (and associated weakness of the model described here) may be due to a more fundamental limitation on the Quantum Kernel Method wherein some circuits map real data points into a Hilbert space representation with vanishing inner products.

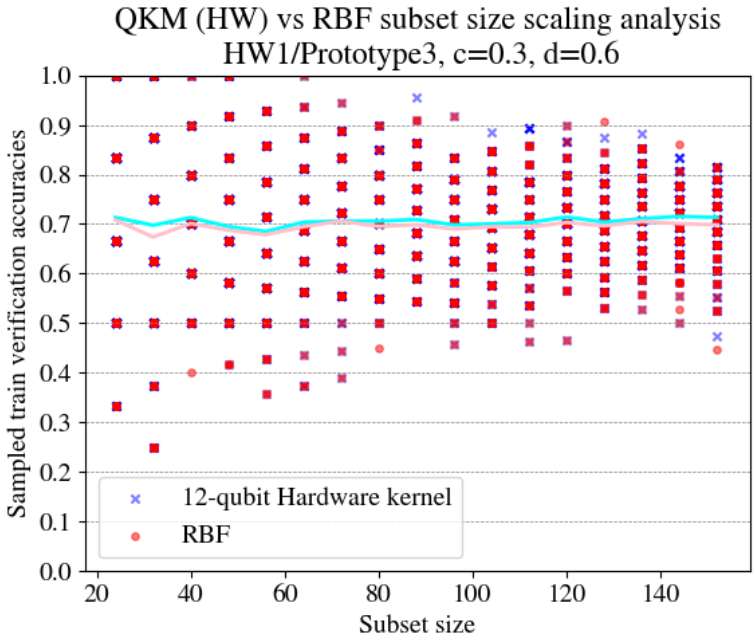


Figure 5.20: SVM classifier performance for 12-qubit PROTO-3 using a 200×200 input Gram matrix over SL training data. For each subset size x shown on the horizontal axis, the blue scatter points show accuracies for SVMs trained on a size $\frac{4}{5}x$ random dataset and tested on the remaining $\frac{1}{5}x$ points. Accuracies assume discrete values according to integral combinations of class sizes for events in the testing subset (i.e. the minimum observable difference in accuracy is $\frac{1}{5}x$). Both RBF and PROTO-3 SVM performances track the class balance boundary of 70%.

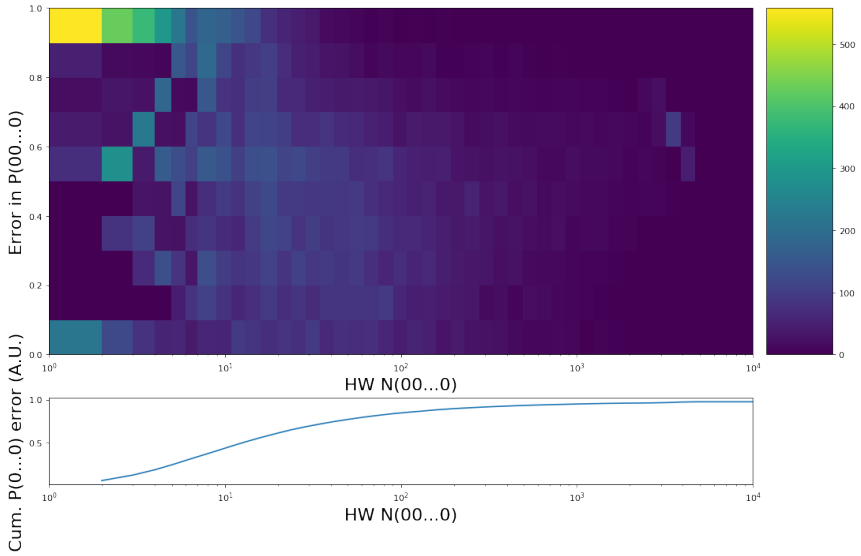


Figure 5.21: The normalized relative error in the HW $P(\underline{0}^{12})$ with respect to SIM $P(\underline{0}^{12})$ values shown on the vertical axis of the 2D histogram is highly concentrated among HW circuits for which the total counts of the all-zeros bitstring $N(\underline{0}^{12}) < 10$ (total counts are captured on the horizontal, logarithmic axis). Normalized relative error is defined here as $\frac{a-b}{max(a,b)}$ for two scalars a, b, which is bounded on $[0, 1]$. The inset histogram shows the sum of all bins over relative error for each bin of $N(\underline{0}^{12})$ normalized by the total cumulative error over all 20100 events. Zero-count events are not included in the 2D histogram but to contribute to the normalization factor for the cumulative relative error.

5.6 Determining the minimum computable kernel element

With the bitflip correction validated from results in 5.3.2, there remains an additional limitation to computing any given kernel element $k(x_i, x_j)$. The large error associated with small kernel entries presented in Section 5.5 indicated a limitation on the 12-qubit circuit architecture, and Section 4.3.3 alluded to the possibility of vanishing inner products (and therefore $k(x_i, x_j)$ values) as a function of the number of qubits in the system. As the expected value of $k(x_i, x_j)$ approaches zero, the number of circuit repetitions needed to extract this value from experiment increases accordingly.

To test this effect, I constructed a dataset of kernel elements by sampling 15 entries from each of 10 neighborhoods spaced evenly over a sorted set of $k(x_i, x_j)$ values from a 400×400 gram matrix with entries computed by simulated *wavefunction inner products* for the PROTO-3 architecture. This subset preserves the median value of $|\langle x_i|x_j\rangle|^2 = P(\underline{0}^n) = k(x_i, x_j)$ over a validated training set and is therefore representative of results expected for computing full gram matrices using the PROTO-3 architecture. In addition, the sampling process was repeated for each of two more circuit architectures in which X gates were sporadically introduced to test the robustness of the bitflip correction algorithm (see Section 5.3.2); all three sets of outcomes are aggregated below with only a modest effect on the subset's median $|\langle x_i|x_j\rangle|^2$ value.

Figures 5.22-5.24 compare the noiseless simulation of exact values for $|\langle x_i|x_j\rangle|^2$ to the corresponding HW outcomes (with statistical noise) over the aggregated subset, for 10000 repetitions on 8, 10, and 12-qubit PROTO-3 circuits respectively (4- and 6-qubit results are provided in Appendix B). The following trends appear

- All HW results fail to accurately predict values for $|\langle x_i|x_j\rangle|^2$ that approach zero.
- For values greater than $n = 8$ qubits, the statistical observable limit for $P(\underline{0}^n)$ begins to affect outcomes. In the case of $n = 10$ qubits (Figure 5.23) 27 events with zero counts on the all-zeros bitstring could be reasonably dropped without affecting bulk trends in the sample. However, this was not the case for $n = 12$ qubits (Figure 5.24), for which roughly a quarter of the entire population of events resulted in $P(\underline{0}^n) = 0$ on hardware.
- The Mean Squared Error (MSE) and Mean Absolute Error (MAE) of $\log(P(\underline{0}^n))$ compared to $\log(|\langle x_i|x_j\rangle|^2)$ steadily grows with respect to the number of qubits (log comparisons were chosen so that the contribution of every datapoint to the mean error is of the same order of magnitude).

- Distributions on $|\langle x_i|x_j\rangle|^2$ are not normal, and exhibit a "tail" towards the lower end with respect to the average inner product for a given number of qubits

Figures 5.25-5.27 show the bitflip-corrected versions of the HW plots in Figures 5.22-5.24. Not only do the trends in outcomes improve, but the trend persists for results from CORR[HW] even for events where $P(\underline{0}^n)$ was at the naive threshold of $1/(\text{number of repetitions}) = 1e-4$ that was directly observable from hardware. In other words, a **single count of the all-zeros bitstring can be corrected to yield an experimental value of** $|\langle x_i|x_j\rangle|^2 \gtrapprox 1e-5$. However from Figure 5.27 it is clear that this is *not the case* for events where $P(\underline{0}^n) = 0$ on hardware: **Corrected outcomes originating from zero prior observed counts were consistently high error**.

The CORR[HW] outcomes indicate a general improvement in MSE and MAE compared to HW alone, and demonstrate that the experiment yields qualitatively good results for values of $k(x_i, x_j)$ in the range $|\langle x_i|x_j\rangle|^2 \gtrapprox 1e-5$, $P_{HW}(\underline{0}^n) = 0 \gtrapprox 1e-5$. Figure 5.28 captures the scaling behavior in these trends by plotting both log-MSE on CORR[HW] versus SIM and the trends in the median magnitude of encoded inner products as a function of $n = 4, 6, 8, 10, 12$ qubits.

The super-exponential trend in log-error for corrected hardware outcomes indicates a large sensitivity to the number of qubits. The almost perfectly exponential decrease in median $|\langle x_i|x_j\rangle|^2$ indicates potential problems for scaling QKM algorithms to larger numbers of qubits in general, and this value crossing minimum observable statistic threshold of $1e-4$ after $n = 10$ qubits provides some explanation for the accelerating behavior of the SIM vs. CORR[HW] error. As alluded to in Section 4.3.3, if a QKM circuit were to map data points to random points in 2^{2n}-dimensional Hilbert space, the average inner product (and therefore value of $k(x_i, x_j)$) between mapped points would scale like $O(1/2^{2n})$ as the dimensionality of the space containing the mapped points quickly outpaced the number of of points. This closely describes the trend in $k(x_i, x_j)$ values for the PROTO-3 circuit, which resulted in the median inner product to sample being much smaller than the minimum value observable on hardware using a 10,000-repetition experiment. Furthermore, given the *exponential* rate of decrease in median inner product values, no reasonable sampling scheme can continue to accurately extract $k(x_i, x_j)$ as n continues to climb beyond 10 qubits.

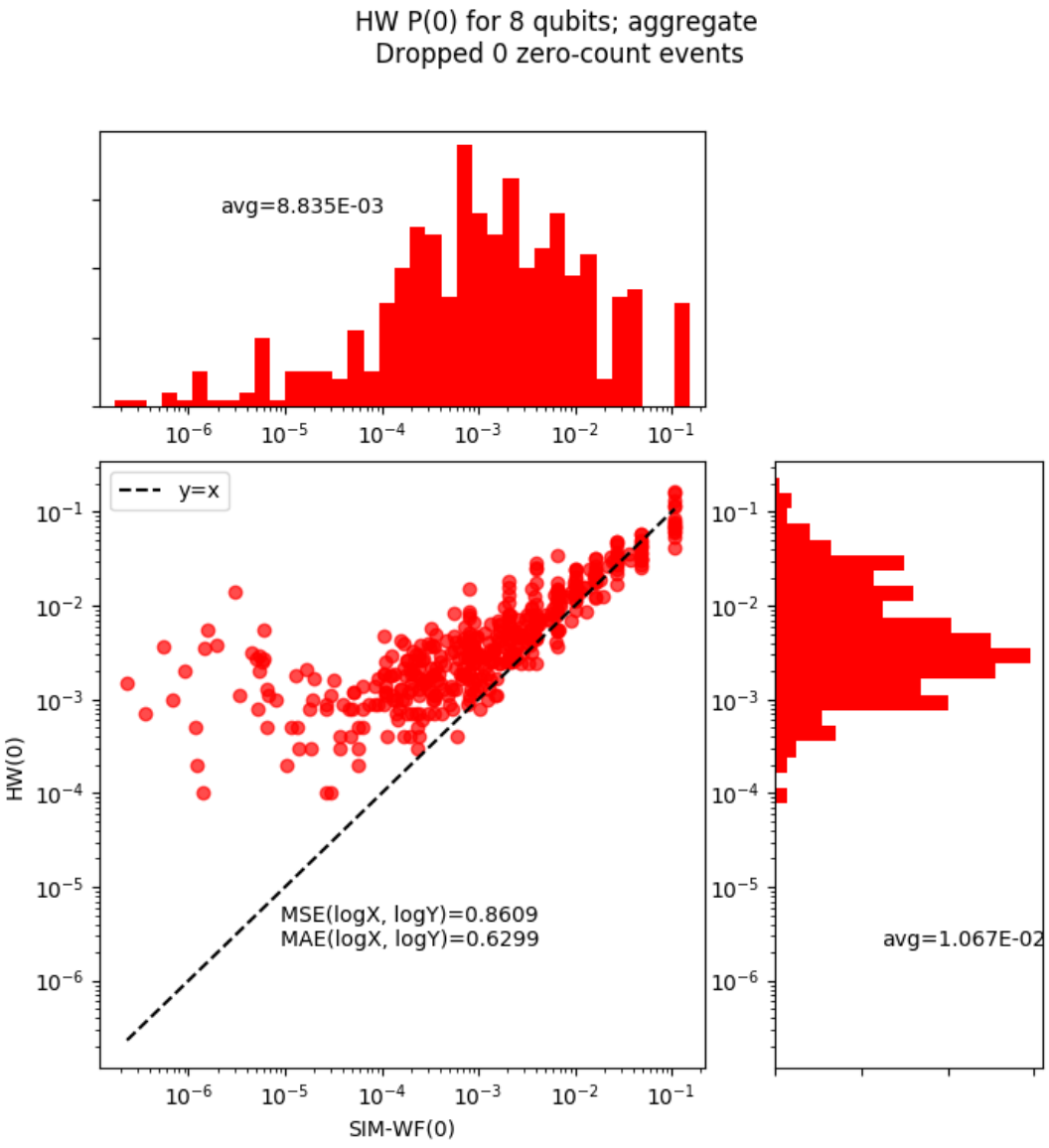


Figure 5.22: Comparison of simulated $|\langle x_i|x_j\rangle|^2 = P(\underline{0}^8)$ values (horizontal axis) to the corresponding HW P(0) (vertical axis) for each (x_i, x_j) pair in the median preserving data subset (see description in main body) shows that the lowest performance in hardware outcomes can be attributed to events associated with smaller wavefunction inner products. In this and all similar plots, there is statistical noise only along the vertical axis corresponding to the size-10000 sample of bitstrings.

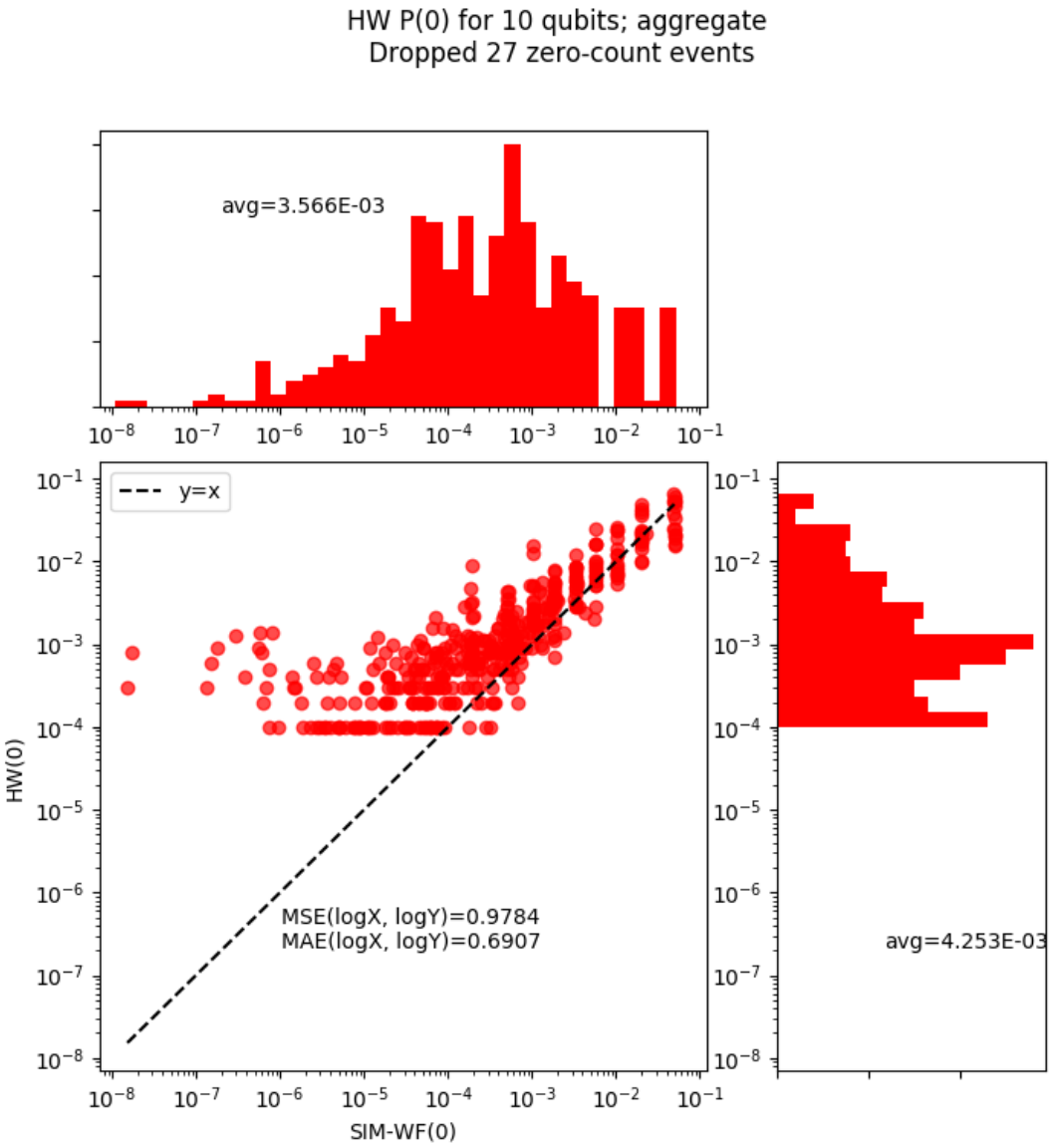


Figure 5.23: To allow use of the Log-MSE error metric for this aggregated dataset, 27 zero-count events were dropped from the set of HW outcomes. At 10 qubits, the minimum observable statistic of 1e-4 for this experiment has the noticeable effect of biasing HW outcomes away from equality to the value predicted by noiseless simulation for $|\langle x_i|x_j\rangle|^2 \lessapprox$ 1e-4.

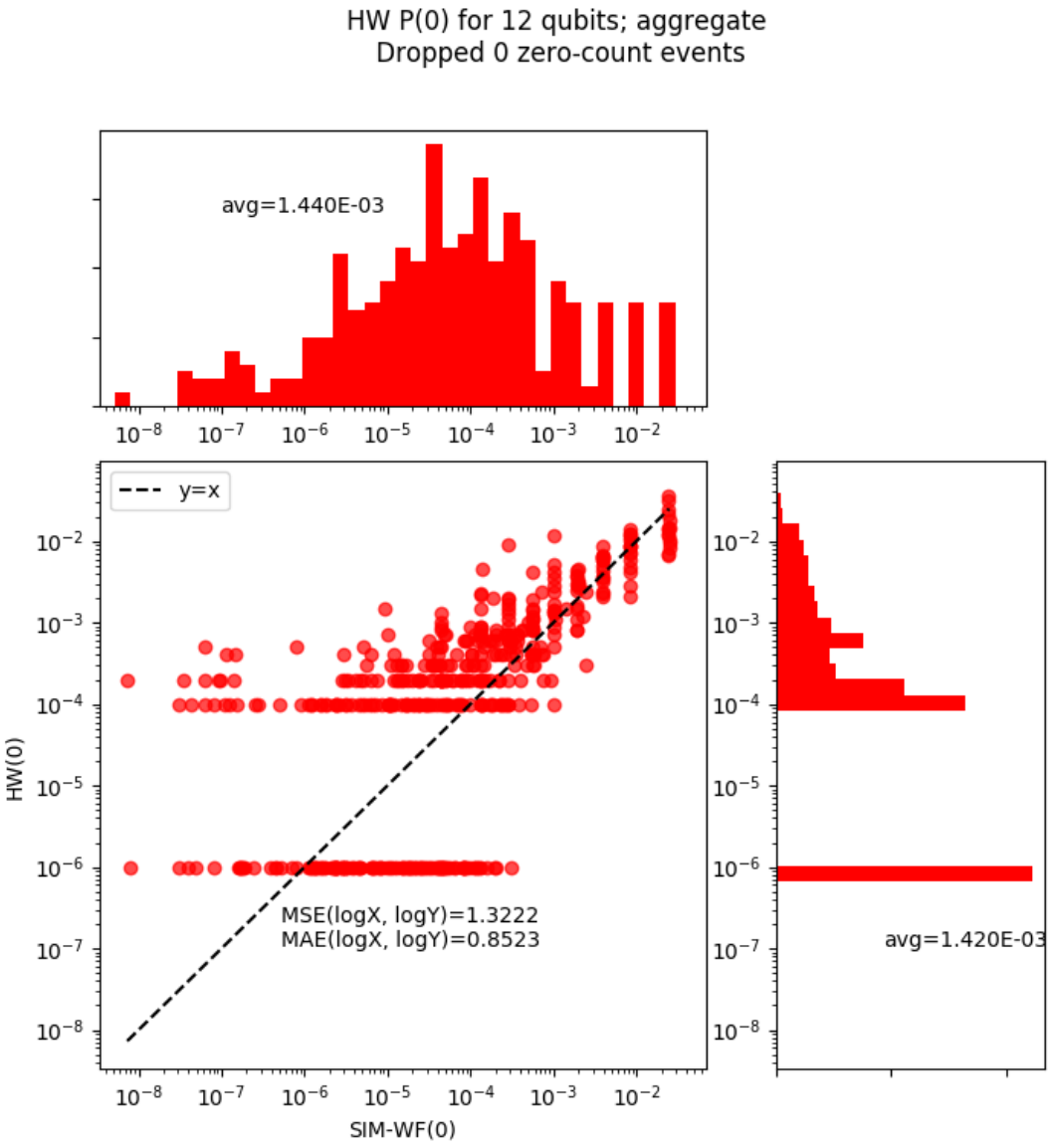


Figure 5.24: 12-qubit results from the median-preserving data subset (see main body) show that uncorrected hardware outcomes are barely correlated with their corresponding noiseless simulation values. The 127 circuits (roughly 1/4 of the dataset) that resulted in zero observed all-zeros bitstrings have been placed in an overflow bin labeled 1e-6 (the log-MSE metrics should therefore be ignored for this plot).

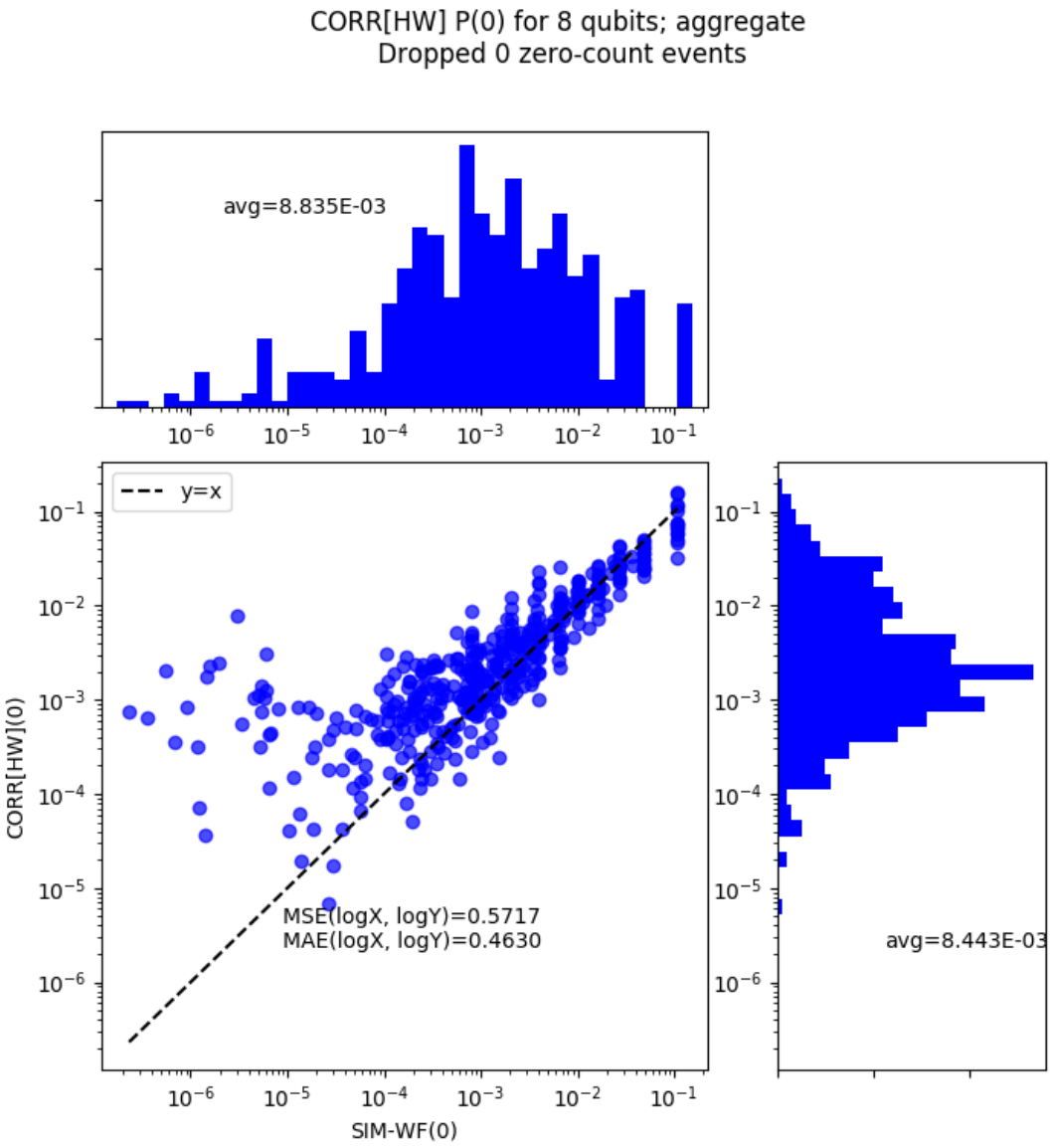


Figure 5.25: This corrected version of Figure 5.22 shows large improvement in $P(\underline{0}^8)$, especially for $k(x_i, x_j)$ values approaching the minimum observable statistic of 1e-4.

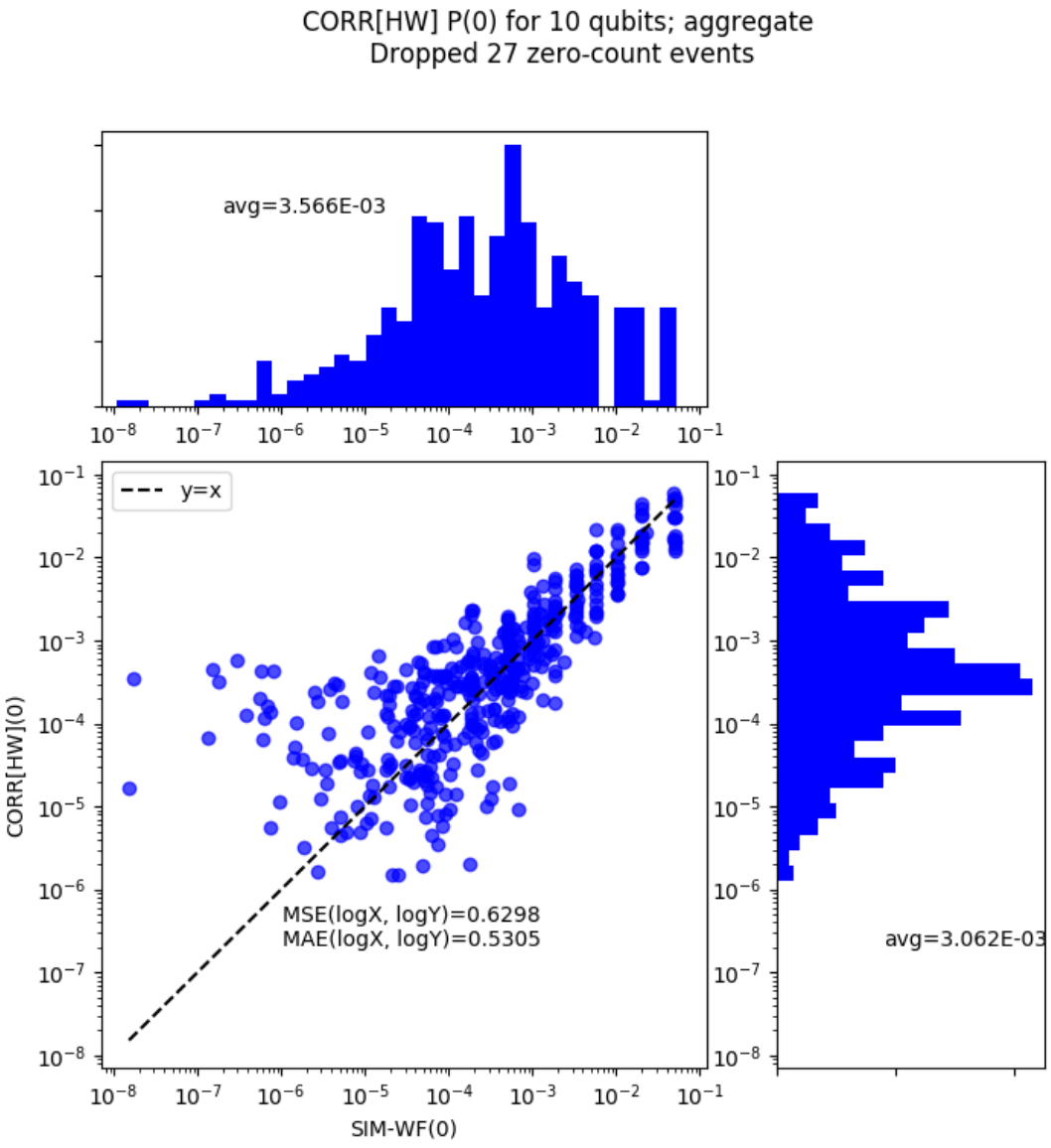


Figure 5.26: The corrections applied to 10-qubit results not only greatly improve trends in agreement between HW and SIM, but demonstrate that the bitflip correction procedure is capable of recovering values for $|\langle x_i|x_j\rangle|^2$ from hardware with magnitudes as low as 1e-5 with reasonable qualitative agreement to simulation. This is an order of magnitude lower than the smallest probability $P(\underline{0}^{10})$ than can actually be observed in hardware for this experiment, suggesting that the bitflip correction may be even more robust than initially shown in Section 5.3.2.

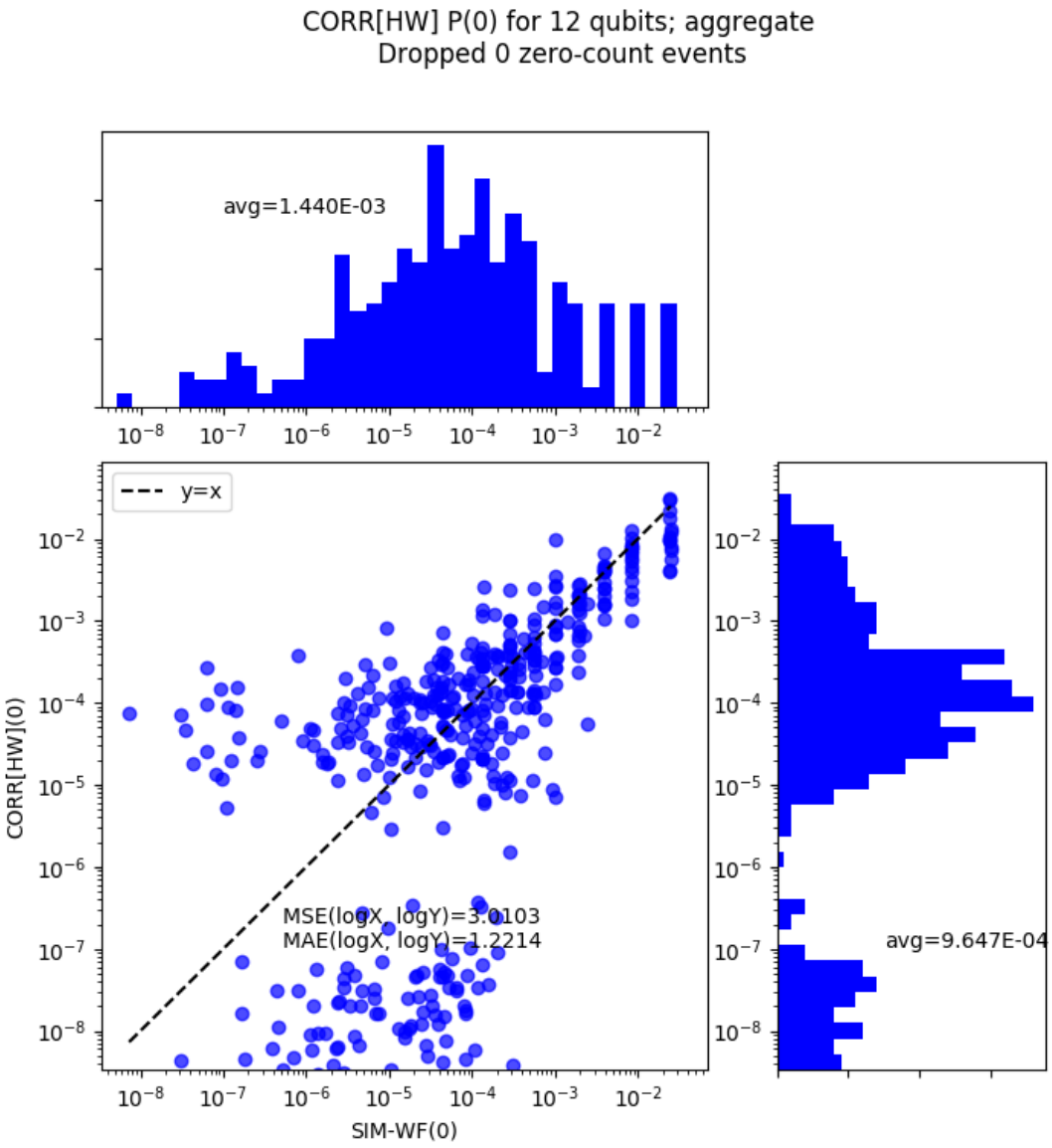


Figure 5.27: A majority of the error in the outcomes for the median $k(x_i, x_j)$-preserving 12 qubit data subset is due to CORR[HW](0) values that were computed using *zero* observed $\underline{0}^{12}$ bitstrings. This demonstrates that, while the bitflip correction algorithm may be able to extract useful information from a single observed all-zeros bitstring, it cannot do so when no all-zeros bitstrings were observed.

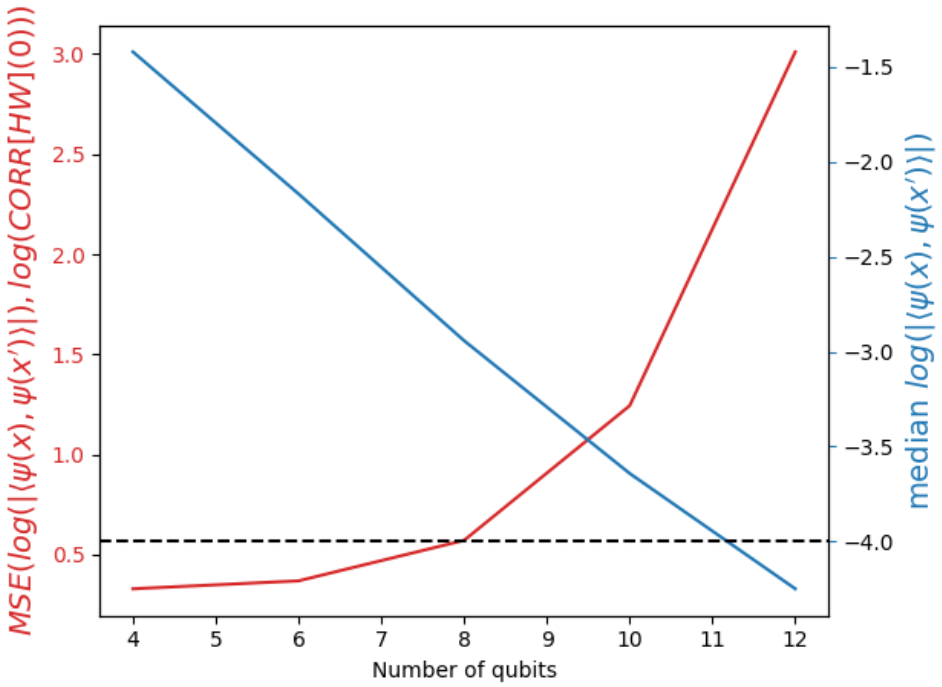


Figure 5.28: Scaling trends for data aggregated from 450 different circuits sampled for 1e4 repetitions provide a hardware-independent explanation for poor hardware behavior above 10 qubits. For greater than 10 qubits, the median inner product to be sampled from $P(\underline{0}^n)$ passes below the minimum observable frequency for 1e4 repetitions; in other words, for > 11 qubits over half of the values of $k(x_i, x_j)$ extracted from hardware will be based on experiments yielding the all zeros bitstring either zero or one time out of ten thousand. The exponential trend in $|\langle x_i|x_j\rangle|^2$ suggests that this cannot be combated by simply increasing the number of samples taken from each circuit.

Chapter 6

Conclusion

Quantum Machine Learning presents exciting prospects for leveraging the power of quantum computers to analyze real-world data by encoding classical data into quantum states for processing. However this potential comes with the burden of validating the outcomes of quantum hardware as well as the performance of the QML algorithms themselves. This thesis set out to provide the means to approach this validation via three distinct goals:

1. Develop a noise model that is predictive of hardware outcomes (Chapter 2).
2. Verify and optimize the outputs of quantum hardware (Chapter 3).
3. Maximize the accuracy of an SVM model that employs QKM (Chapters 4-5).

The rudimentary noise model suggested in this thesis proved to be effective for predicting the outcomes of circuits that ran on real hardware on $\lessapprox 8$ qubits with depth $\lessapprox 25$ gates, while at the same time providing a valuable tool for validating the performance of the Bayesian iterative unfolding bitflip correction method. Equally important was the finding that the noise model *fails* to capture the dynamics of the hardware for large numbers of qubits which indicates either a degree of inaccuracy in accepted gate fidelity metrics as Kraus operator parameters, or a large contribution of non-local/non-Markovian sources of error to many-qubit hardware runs, or both.

In the course of optimizing and postprocesing hardware outcomes, this thesis provided foundations for implementing automated qubit selection and CPMG-style T_2 optimizations to reduce the impact of noise on algorithms designed for NISQ devices. Crosstalk diagnostics demonstrated one possible technique for mitigating crosstalk error (assumed to be

a large source of error for the larger circuits implemented in this work) but also demonstrated that crosstalk error induced by massively parallel gate execution is preferable to erro introduced by any sequential execution of the same set of gates.

This thesis provided evidence for a 10 qubit limit on QKM circuit size that was due in part to hardware noise, but also due to an uncovered exponential scaling in magnitudes of $|\langle\phi(x)|\phi(x')\rangle|^2$ over Hilbert space embeddings of classical data $x, x' \in \mathbb{R}^d$. This result may indicate a fundamental limitation on the advantage that QKM-based classifiers can bring to classical machine learning settings.

Regardless of the prospects of quantum kernel methods in the long run, understanding, optimizing, and iterating algorithm design based on hardware outcomes from NISQ computers will continue to be challenge for algorithms of all types, and the results provided here contribute a few more strategies to approach this problem with.

Future Work

All of the results in this thesis represent a work in progress using a prototype quantum computer and a fairly novel QML algorithm for which the benefits of using are still largely unproven. As such, it has provided a starting point for many potentially fruitful investigations to be continued as understanding of both the QKM algorithm and the quantum hardware improve.

Section 3.2.3 demonstrated that QPT could be used to perform detailed analysis of non Markovian noise channels, which could both further our understanding of crosstalk noise on superconducting qubit devices as well as provide a means of correcting unitary components of these channels. The performance of the noise model might be reasonably improved by converting parallelized QPT results for $\sqrt{\text{iSWAP}}$ gates into noise channels compatible with our circuit model noise simulation via Equation 2.4.

Sections 5.2.1 and 5.1.2 introduced the difficulty of engineering both a QKM circuit that would perform well in an SVM with respect to a specific dataset as well as a subset of datapoints for which the performance of the SVM could be trusted. While dataset validation is a well known problem in classical machine learning, future work on this project will be devoted to integrating these two processes into a single process for prototyping circuit architectures, perhaps using reinforcement learning or some other classical machine learning algorithm to learn what families of QKM circuits are effective for a given dataset.

Sections 3.2.4 and 5.2.3 introduced two hardware optimizations that have the potential to significantly improve results extracted from this project's hardware and NISQ devices

in general. While promising, these techniques will need to be rigorously tested in different contexts to determine the extent of their efficacy and whether their performance enhancements generalize.

Finally, Sections 5.3.2 and 5.4 introduced and provided evidence for the performance of a bitflip correction scheme based on Bayesian iterative unfolding over a Hamming-truncated subspace of bitstring outcomes. This work's unique reliance on the all-zero's bitstring to compute kernel elements gives (Hamming-truncated) bitflip correction a central role in analyzing and interpreting results for the QKM. This work provided some initial investigation into the performance of this bitflip correction method, and future work will follow up with broader analyses on whether or not the observed behavior can be generalized.

References

[1] Bologna Lens Factory.

[2] N Aronszajn. Theory of Reproducing Kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G.S.L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. *Quantum supremacy using a programmable superconducting processor*, volume 574. 2019.

[4] Jeremy Bejanin. Schrodinger.jl.

[5] Alexandre Blais, Ren Shou Huang, Andreas Wallraff, S M Girvin, and R J Schoelkopf. Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation. *Physical Review A - Atomic, Molecular, and Optical Physics*, 69(6):1–14, 2004.

[6] F. Bloch. Nuclear induction. *Physical Review*, 70(7-8):460–474, 1946.

[7] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, 2018.

[8] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, 2018.

[9] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. Training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[10] James Bourbeau and Zigfried Hampel-Arias. PyUnfold: A Python package for iterative unfolding. *Journal of Open Source Software*, 3(26):741, 2018.

[11] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, Murphy Yuezhen Niu, Ramin Halavati, Evan Peters, Martin Leib, Andrea Skolik, Michael Streif, David Von Dollen, Jarrod R. McClean, Sergio Boixo, Dave Bacon, Alan K. Ho, Hartmut Neven, and Masoud Mohseni. TensorFlow Quantum: A Software Framework for Quantum Machine Learning. pages 1–39, 2020.

[12] H. Y. Carr and E. M. Purcell. Effects of diffusion on free precession in nuclear magnetic resonance experiments. *Physical Review*, 94(3):630–638, 1954.

[13] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, 2019.

[14] G. D'Agostini. A multidimensional unfolding method based on Bayes' theorem. *Nuclear Inst. and Methods in Physics Research, A*, 362(2-3):487–498, 1995.

[15] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A Quantum Approximate Optimization Algorithm. pages 1–16, 2014.

[16] Edward Farhi and Hartmut Neven. Classification with Quantum Neural Networks on Near Term Processors. https://arxiv.org/abs/1802.06002.

[17] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982.

[18] J Friedman. Flexible Metric Nearest Neighbor Classification. 1994.

[19] Lov K Grover. A Fast Quantum Mechanical Algorithm for Database Search. pages 212–219, 1996.

[20] A A Hagberg, D A Schult, and P J Swart. Exploring network structure, dynamics, and function using NetworkX. *7th Python in Science Conference (SciPy 2008)*, (SciPy):11–15, 2008.

[21] E.L. Hahn. Spin Echoes. *Phys. Rev.*, 80(4):580, 1950.

[22] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019.

[23] Michał Horodecki, Paweł Horodecki, and Ryszard Horodecki. General teleportation channel, singlet fraction, and quasidistillation. *Physical Review A - Atomic, Molecular, and Optical Physics*, 60(3):1888–1898, 1999.

[24] Evan Jeffrey, Daniel Sank, J Y Mutus, T C White, J Kelly, R Barends, Y Chen, Z Chen, B Chiaro, A Dunsworth, A Megrant, P J O'Malley, C Neill, P Roushan, A Vainsencher, J Wenner, A N Cleland, and John M Martinis. Fast accurate state measurement with superconducting qubits. *Physical Review Letters*, 112(19):1–5, 2014.

[25] Nathaniel Johnston and David W. Kribs. Quantum gate fidelity in terms of Choi matrices. *Journal of Physics A: Mathematical and Theoretical*, 44(49):1–16, 2011.

[26] George C Knee, Eliot Bolduc, Jonathan Leach, and Erik M Gauger. PHYSICAL REVIEW A 98 , 062336 (2018) Quantum process tomography via completely positive and trace-preserving projection. 062336:1–11, 2018.

[27] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland. Randomized benchmarking of quantum gates. *Physical Review A - Atomic, Molecular, and Optical Physics*, 77(1), 2008.

[28] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning. pages 1–11, 2020.

[29] Andrea Mari, Thomas R. Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. pages 1–12, 2019.

[30] Jarrod R. McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2), 2016.

[31] S. Meiboom and D. Gill. Modified Spin-Echo Method for Measuring Nuclear Relaxation Times. *Review of Scientific Instruments*, 29(8):688–691, 1958.

[32] Seth T. Merkel, Jay M. Gambetta, John A. Smolin, Stefano Poletto, Antonio D. Córcoles, Blake R. Johnson, Colm A. Ryan, and Matthias Steffen. Self-consistent quantum process tomography. *Physical Review A - Atomic, Molecular, and Optical Physics*, 87(6):1–10, 2013.

[33] C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, S. V. Isakov, V. Smelyanskiy, A. Megrant, B. Chiaro, A. Dunsworth, K. Arya, R. Barends, B. Burkett, Y. Chen, Z. Chen, A. Fowler, B. Foxen, M. Giustina, R. Graff, E. Jeffrey, T. Huang, J. Kelly, P. Klimov, E. Lucero, J. Mutus, M. Neeley, C. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, H. Neven, and J. M. Martinis. A blueprint for demonstrating quantum supremacy with superconducting qubits. *Science*, 360(6385):195–199, 2018.

[34] Andrew Ng. CS229 Lecture notes Margins : SVM. *Intelligent Systems and their Applications IEEE*, pt.1(x):1–25, 2012.

[35] Michael A. Nielsen. A simple formula for the average gate fidelity of a quantum dynamical operation. *Physics Letters, Section A: General, Atomic and Solid State Physics*, 303(4):249–252, 2002.

[36] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information 10th Anniversary Edition*. 2005.

[37] J. L. O'Brien, G. J. Pryde, A. Gilchrist, D. F.V. James, N. K. Langford, T. C. Ralph, and A. G. White. Quantum process tomography of a controlled-NOT gate. *Physical Review Letters*, 93(8):1–4, 2004.

[38] P. J. J. O'Malley. Superconducting Qubits: Dephasing and Quantum Chemistry. *Thesis*, (June), 2016.

[39] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[40] John Preskill. Fault-tolerant quantum computation. pages 1–58, 1997.

[41] John Preskill. Lecture Notes for Physics 229: Quantum Information and Computation. 1998.

[42] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2(July):79, 2018.

[43] Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. *46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561, 2008.

[44] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference*, (1):1–8, 2009.

[45] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*, 1(1):1313–1320, 2009.

[46] Norman F Ramsey. A Molecular Beam Resonance Method with Separated Oscillating Fields. *Phys. Rev.*, 78(6):695, 1950.

[47] Martin Rötteler. Quantum algorithms for highly non-linear Boolean functions. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, 2010.

[48] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2111:416–426, 2001.

[49] Maria Schuld and Nathan Killoran. Quantum Machine Learning in Feature Hilbert Spaces. *Physical Review Letters*, 122(4), 2019.

[50] Robert Sedgewick. *Algorithms in c, Part 5: Graph Algorithms, Third Edition*. Addison-Wesley Professional, third edition, 2001.

[51] Peter W Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *arXiv*, (9508027v2 25):124–134, aug 1995.

[52] Alex J Smola and Bernhard Scholkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[53] G Wahba and G Kimeldorf. Some Results on Tchebycheffian Spline Functions. *Journal of Mathematical Analysis and Applications*, 33:82–95, 1971.

[54] John Watrous. The Theory of Quantum Information. *The Theory of Quantum Information*, 2018.

[55] C. M. Wilson, J. S. Otterbach, N. Tezak, R. S. Smith, A. M. Polloreno, Peter J. Karalekas, S. Heidel, M. Sohaib Alam, G. E. Crooks, and M. P. da Silva. Quantum Kitchen Sinks: An algorithm for machine learning on near-term quantum computers. pages 1–9, 2018.

[56] Svante Wold, Kim Esbensen, and Paul Geladi. Principal Component Analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[57] M M Wolf. Quantum channels & operations: Guided tour. *Lecture notes available at http://www-m5 ma tum ...*, 2012.

APPENDICES

Appendix A

Calibration data example plots

The results of each job submission to the hardware were returned with a suite of calibration data describing the most recent characterizations of hardware performance. These results are subject to drift over time and therefore comparisons between the noisy simulation and hardware results are sensitive to the accuracy of the calibration data provided to the noise model.

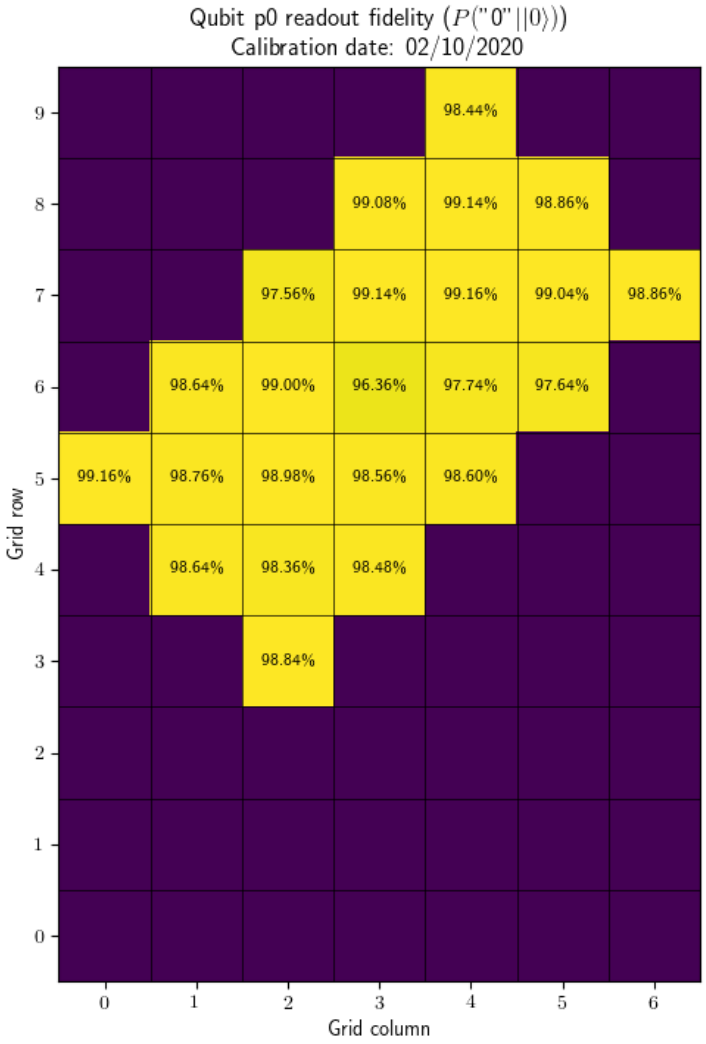


Figure A.1: Sample p_0 values for the 02/10/2020 job submission

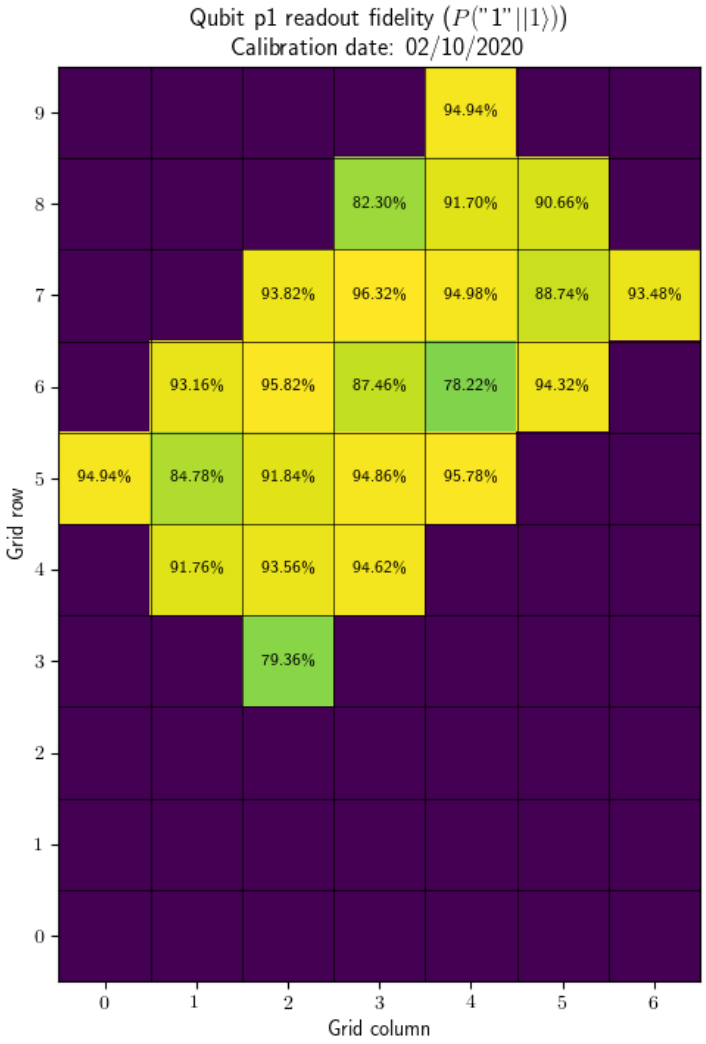


Figure A.2: Sample p_1 values for the 02/10/2020 job submission

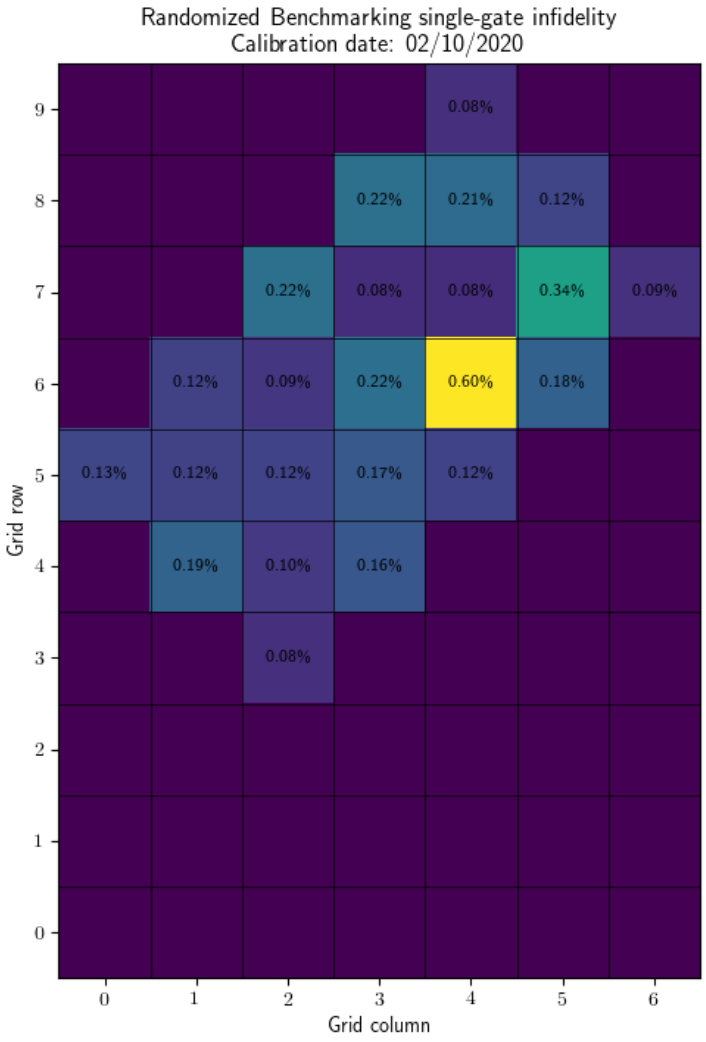


Figure A.3: Sample single-qubit RB gate fidelity values for the 02/10/2020 job submission

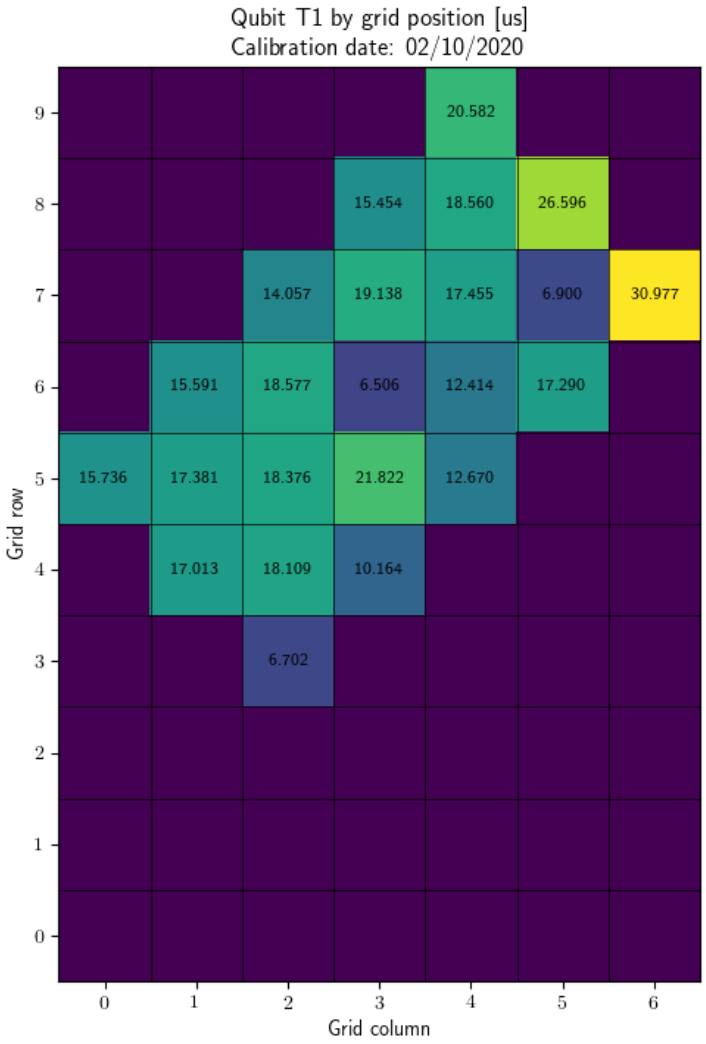


Figure A.4: Sample T_1 values for the 02/10/2020 job submission

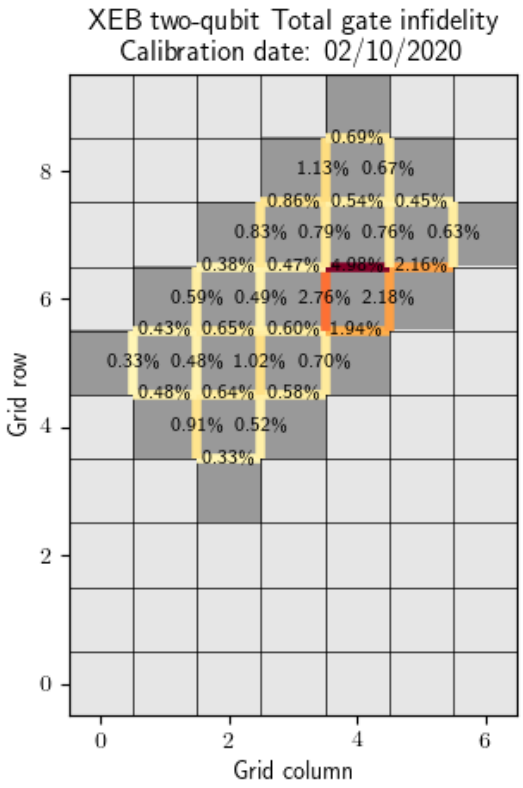


Figure A.5: Sample two qubit gate total XEB fidelity values for the 02/10/2020 job submission

Appendix B

Inner product scaling analysis results for 4 and 6 qubits

The following plots report the 4- and 6-qubit outcomes supporting the scaling analysis developed in Section 5.6.

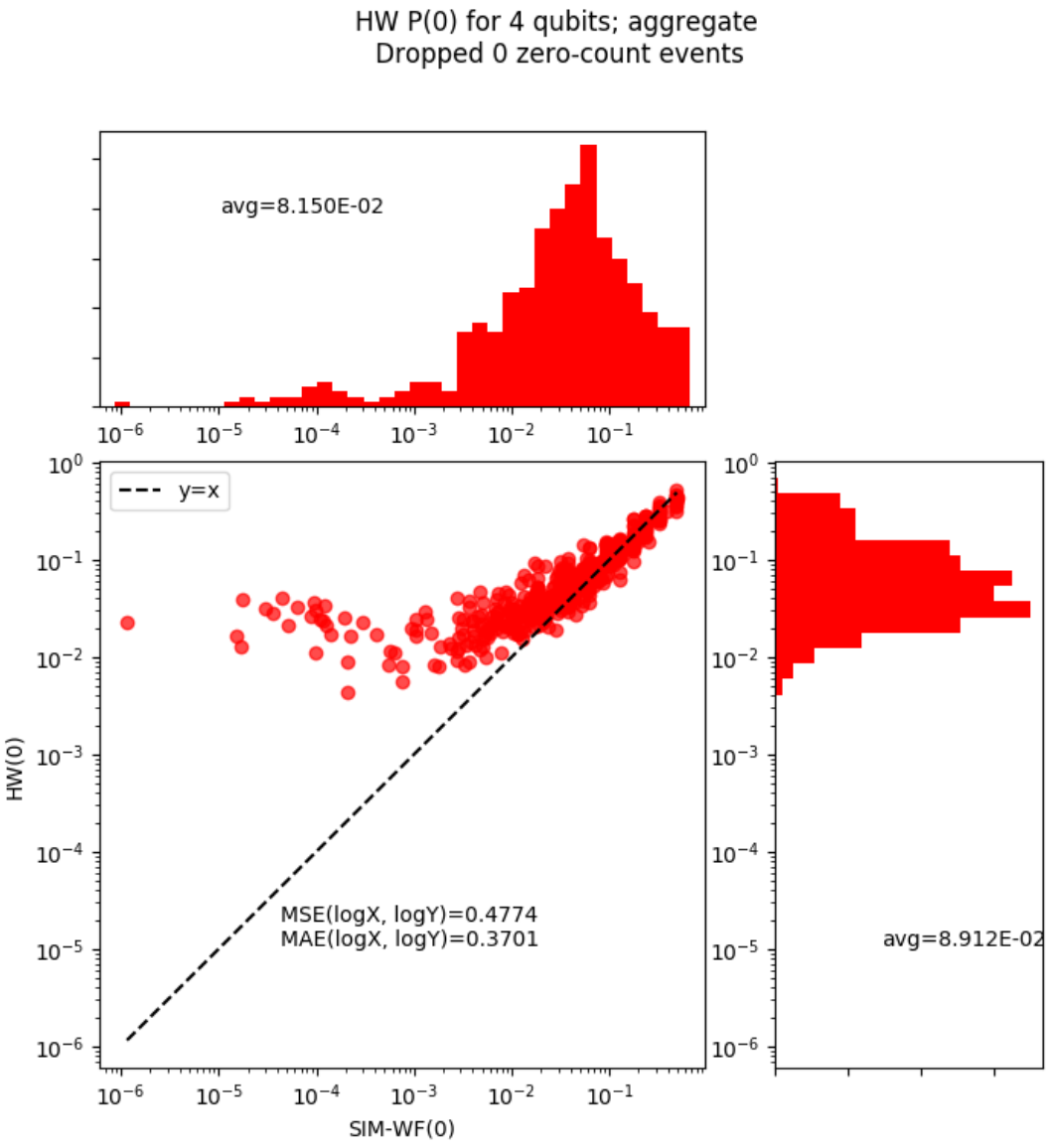


Figure B.1: Aggregated 4 qubit results for PROTO-3, HW vs SIM: Comparison of simulated $|\langle x_i|x_j\rangle|^2 = P(\underline{0}^8)$ values (horizontal axis) to the corresponding HW P(0) (vertical axis) for each (x_i, x_j) pair in the median preserving data subset (see description in main body of Section 5.6). The upwards bias affecting small-magnitude HW P(0) values is especially pronounced for small numbers of qubits, though its not clear why this should be the case.

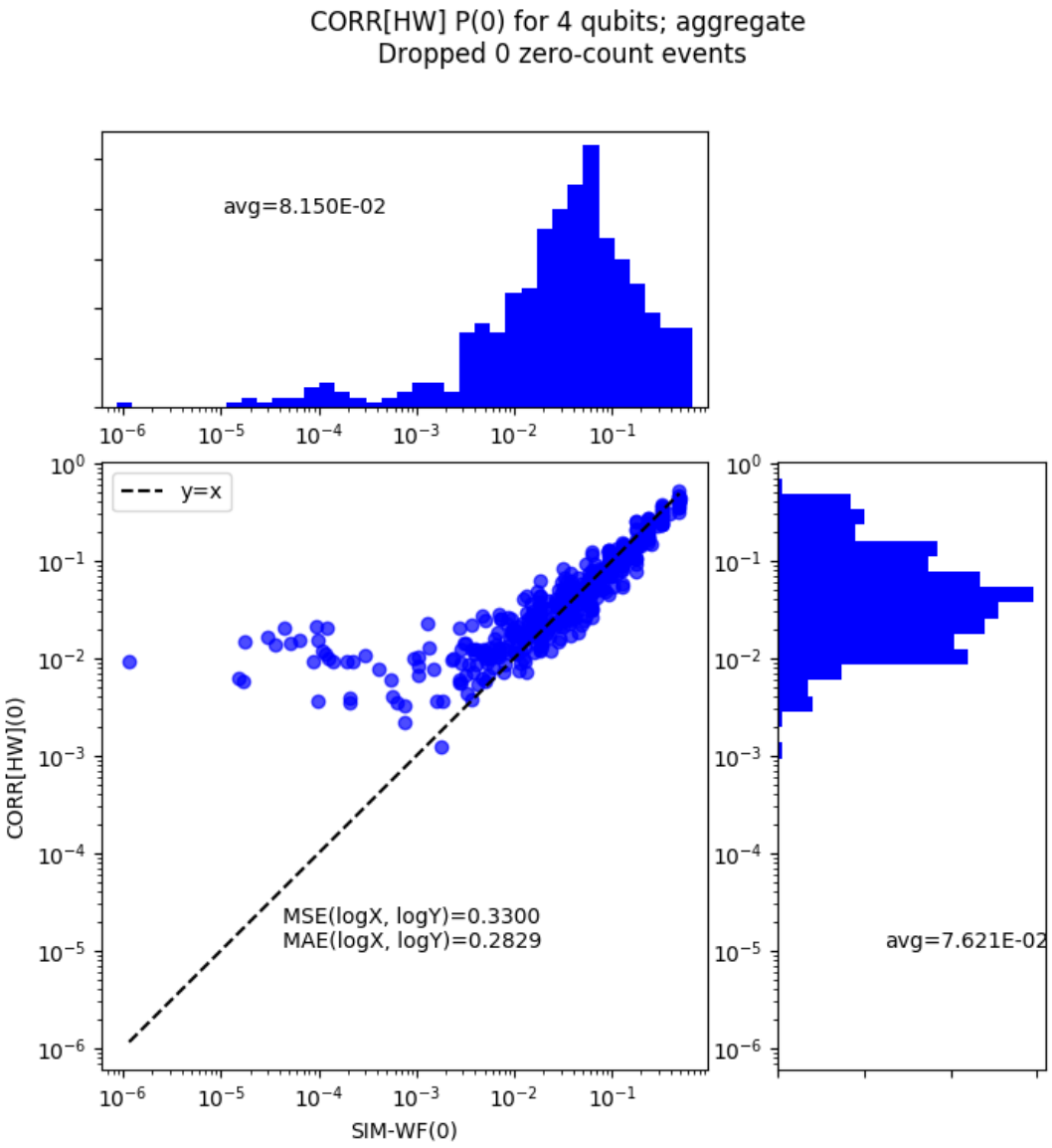


Figure B.2: Aggregated 4 qubit results for PROTO-3, CORR[HW] vs SIM: One of the weaknesses of the bitflip correction algorithm is its difficulty in correcting probabilities over wide ranges. As depicted here, in the presence of heavy bias over a five decade range of probabilities the correction is only effective for around 3 decades of inner product magnitudes.

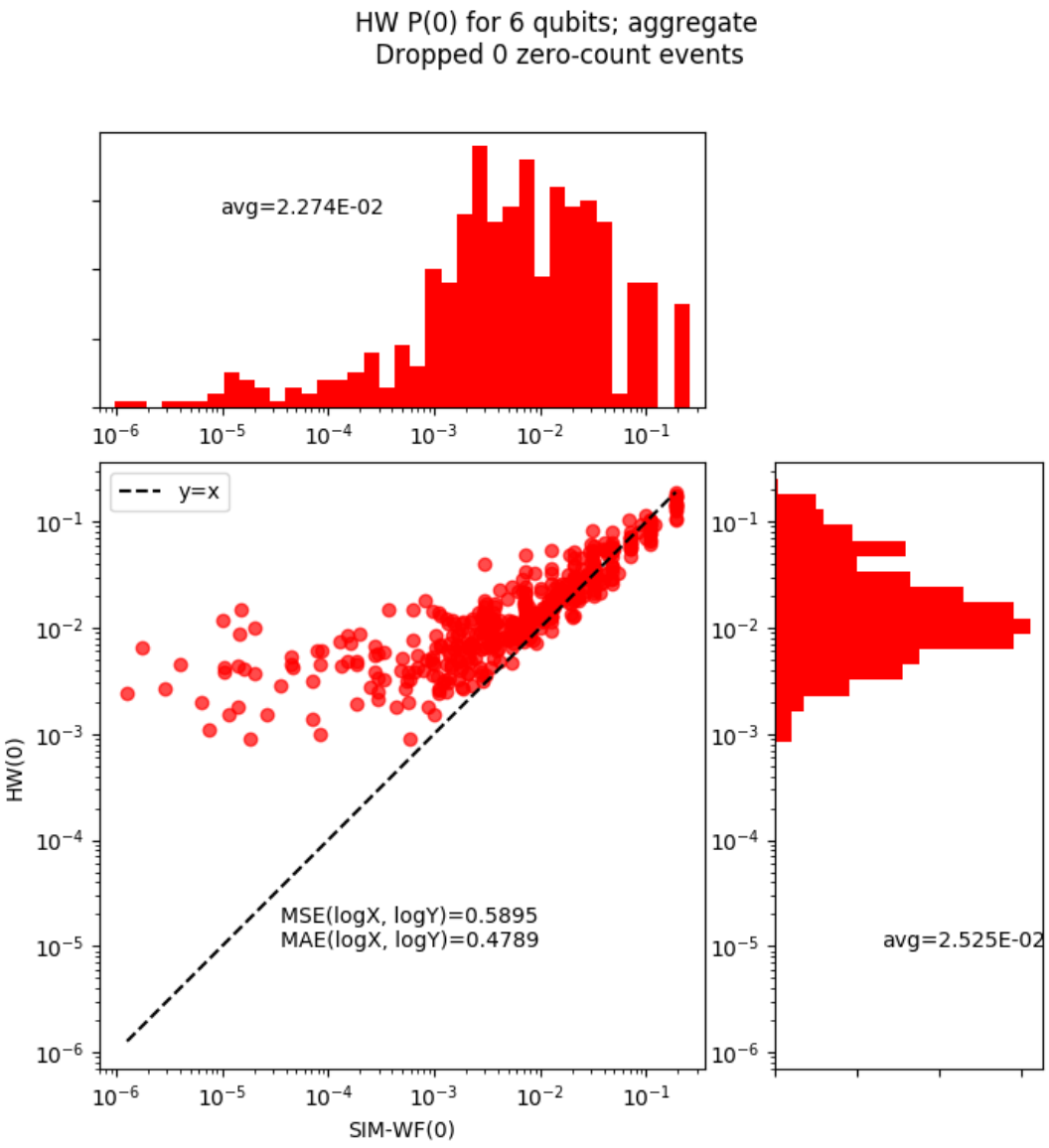


Figure B.3: Aggregated 6 qubit results for PROTO-3, HW vs SIM

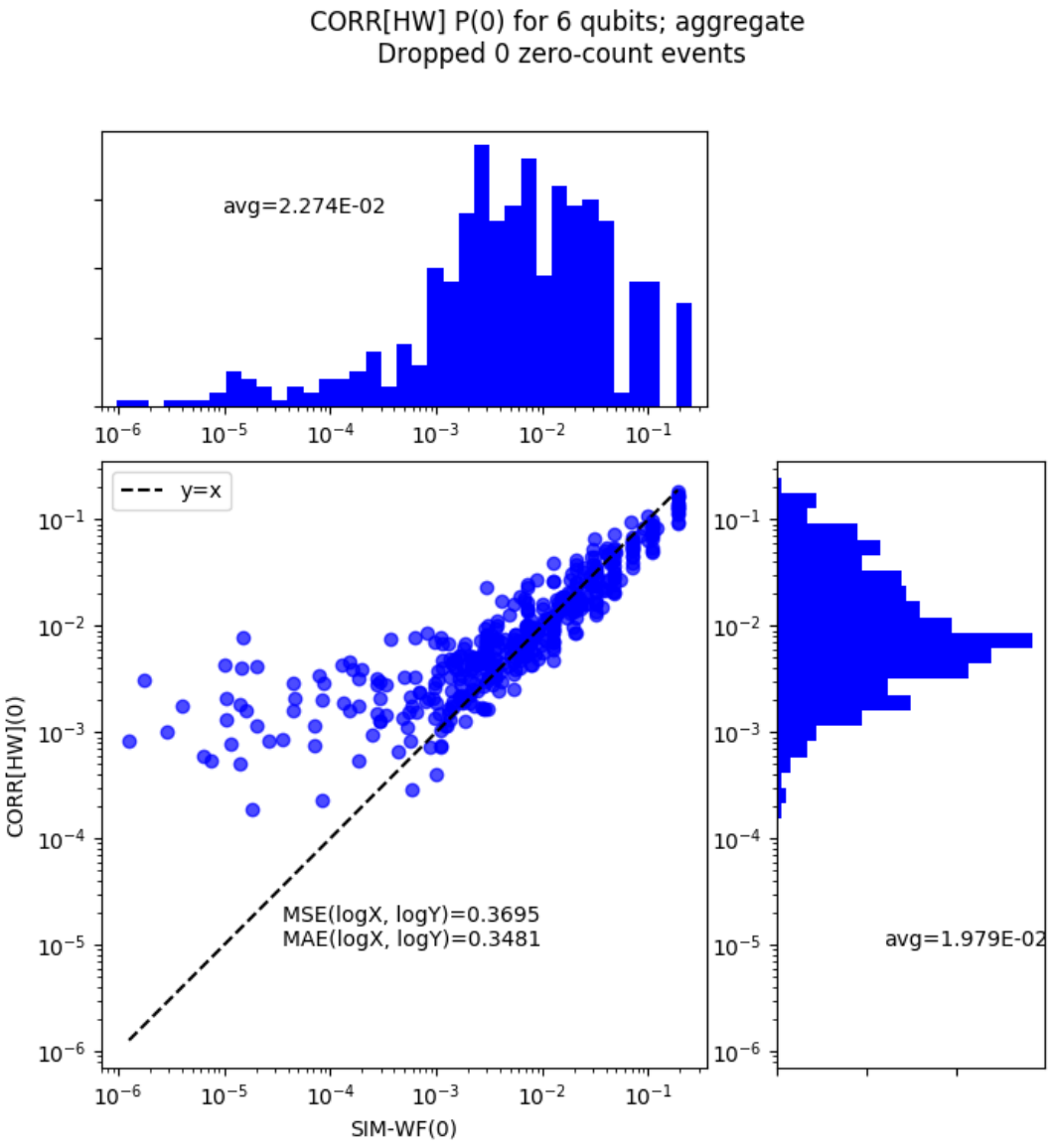


Figure B.4: Aggregated 6 qubit results for PROTO-3, CORR[HW] vs SIM

Glossary

NISQ "Noisy Intermediate Scale Quantum" – A term used to describe early quantum computers that are limited by decoherence and contain only hundreds of qubits. 2, 87

QKM "Quantum Kernel Method" – This describes the general approach of using the Hilbert space of states prepared by a quantum circuit as a feature space for embedding classical data. 4, 6, 46, 68, 87, 88

QML "Quantum Machine Learning" – An umbrella term describing the use of a QPU to enhance the performance of ML models or the application of ML to improve the performance of quantum circuits. 4, 87

QPT "Quantum Process Tomography" – A family of diagnostics used to determine the form of a quantum channel, typically in the context of applying a gate with nonzero error. 12, 29, 32, 88

RB "Randomized Benchmarking" – A gate fidelity diagnostic [27] that captures the average infidelity in sequences of Pauli gates. 13, 17, 19

RBF "Radial Basis Function" – A popular, parametrized kernel with the form $K(x, y) = \exp(-\gamma||\vec{x} - \vec{y}||^2)$ 45, 51

SVM "Support Vector Machine" – A supervised learning algorithm capable of computing linear class boundaries between high-dimensional mappings of data points, corresponding to nonlinear boundaries in the space of the original data. 4, 38, 88

XEB "Cross Entropy Benchmarking" – A gate fidelity diagnostic [7] that summarizes a pseudo-distance between an observed distribution and the Porter Thomas distribution (developed specifically for compatibility with Google's quantum supremacy experiment [3]). 13, 17, 19, 32