# Software Mitigation of Crosstalk on Noisy Intermediate-Scale Quantum Computers

Prakash Murali David C. McKay* Margaret Martonosi Ali Javadi-Abhari*
Princeton University *IBM T. J. Watson Research Center

## Abstract

Crosstalk is a major source of noise in Noisy Intermediate-Scale Quantum (NISQ) systems and is a fundamental challenge for hardware design. When multiple instructions are executed in parallel, crosstalk between the instructions can corrupt the quantum state and lead to incorrect program execution. Our goal is to mitigate the application impact of crosstalk noise through software techniques. This requires (i) accurate characterization of hardware crosstalk, and (ii) intelligent instruction scheduling to serialize the affected operations. Since crosstalk characterization is computationally expensive, we develop optimizations which reduce the characterization overhead. On three 20-qubit IBMQ systems, we demonstrate two orders of magnitude reduction in characterization time (compute time on the QC device) compared to all-pairs crosstalk measurements. Informed by these characterization, we develop a scheduler that judiciously serializes high crosstalk instructions balancing the need to mitigate crosstalk and exponential decoherence errors from serialization. On real-system runs on three IBMQ systems, our scheduler improves the error rate of application circuits by up to 5.6x, compared to the IBM instruction scheduler and offers near-optimal crosstalk mitigation in practice.

In a broader picture, the difficulty of mitigating crosstalk has recently driven QC vendors to move towards sparser qubit connectivity or disabling nearby operations entirely in hardware, which can be detrimental to performance. Our work makes the case for software mitigation of crosstalk errors.

***CCS Concepts*** **• Hardware → Hardware reliability screening; Quantum computation; Hardware reliability screening; Quantum computation; • Software and its engineering → Compilers; Compilers;**

***Keywords*** quantum computing, crosstalk, compiler optimization

**ACM Reference format:**
Prakash Murali, David C. McKay, Margaret Martonosi, Ali Javadi-Abhari. 2020. Software Mitigation of Crosstalk on Noisy Intermediate-Scale Quantum Computers . In *Proceedings of Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16–20, 2020 (ASPLOS '20)*, 16 pages.
https://doi.org/10.1145/3373376.3378477

## 1 Introduction

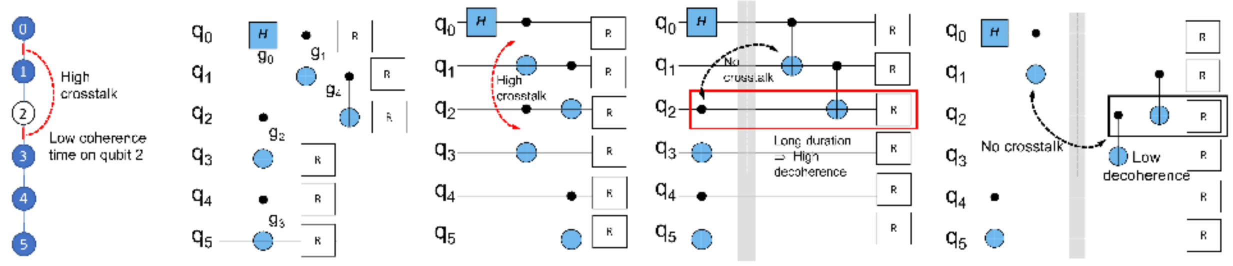
Quantum computing (QC) is a radically new paradigm of computing, where information is stored in *qubits* (quantum bits) and manipulated using instructions known as *gates*. By using quantum effects to efficiently navigate an exponentially-scaling state space, QC systems can arrive at a solution much faster for certain classically-intractable problems. This computational model has proven effective in diverse fields such as cryptography [54], chemistry [28, 48] and machine learning [3, 32].

QC hardware has progressed considerably in the last few years. Prototype systems with 5-20 qubits are now available for broad public use [23] and larger systems with 49-72 qubits are under development or test [17, 22, 26]. The term Noisy Intermediate-Scale Quantum (NISQ) refers to these near-term quantum systems with up to a few hundred qubits and imperfect qubits, gates and readout [49]. While too small and noisy to run large applications, they can support small application studies and are an important milestone on the way to practical QC. Compiler toolflows which optimize programs to make the best use of scarce hardware resources and mitigate the effects of hardware noise are therefore critical for useful computation on NISQ-era devices.

*Crosstalk* is a major source of noise in NISQ systems which corrupts the quantum state when multiple gates (instructions) are executed simultaneously. Crosstalk arises from fundamental challenges in QC hardware design such as unwanted interactions between the qubits and from leakage of the control signals (used to operate the gate) onto qubits which are not part of the intended gate operation. Crosstalk noise is prevalent across many of the leading qubits including superconducting and trapped ion qubits [12, 20, 31, 46, 50]. This paper focuses on crosstalk mitigation in superconducting systems from IBM. Current IBM systems have gate error rates of 1-2% per two-qubit operation [44]; when affected

*ASPLOS '20, March 16–20, 2020, Lausanne, Switzerland*

ACM ISBN 978-1-4503-7102-5/20/03...$15.00
https://doi.org/10.1145/3373376.3378477



(a) Machine (b) Program IR (c) Original Default Schedule (d) High decoherence schedule (e) Desired Schedule

**Figure 1.** (a) An example 6-qubit system. Nodes are physical qubits, and edges are possible CNOT gates. When a CNOT is executed on qubits (0, 1) and another CNOT is executed simultaneously on qubits (2, 3), the error rate of both CNOTs increases because of crosstalk. Qubit 2 has low coherence, which means that long computation on that qubit (including any idle time after the first operation) is highly error prone. (b) An example program IR with parallelized operations. Dangling XOR operations are CNOTs and R is for readout. Time goes left to right. (c) Default schedule for this program on IBM hardware — the schedule maximizes instruction parallelism, but the hardware is restricted to perform all readouts at the same time, so by default, all gates are right-aligned by the hardware scheduler. This schedule suffers from high crosstalk errors. (d) A schedule where the high crosstalk operations are naively serialized, leading to high decoherence error on qubit 2. (e) The desired schedule which avoids high crosstalk and high decoherence errors.

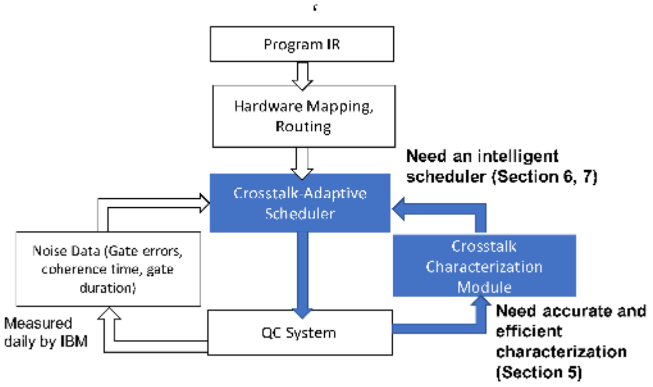

Program IR
Hardware Mapping, Routing
Crosstalk-Adaptive Scheduler
Need an intelligent scheduler (Section 6, 7)
Noise Data (Gate errors, coherence time, gate duration)
Crosstalk Characterization Module
Measured daily by IBM
QC System
Need accurate and efficient characterization (Section 5)

**Figure 2.** Our crosstalk mitigation approach. We develop two building blocks necessary for software mitigation of crosstalk. The first module performs fast and accurate characterization of the crosstalk noise present in the hardware. The second module performs instruction scheduling using the characterization data. Our scheduler serializes high crosstalk instructions but also balances the need to avoid exponential decoherence errors from serialization.

by crosstalk noise, we observe that gate errors can worsen by an order of magnitude. Through an extensive study, we show that these crosstalk effects can significantly impact the total program reliability. Our goal is to mitigate this error in software through intelligent instruction scheduling.

Figure 1 shows the error tradeoffs that influence scheduling decisions. When a pair of simultaneous program operations has high crosstalk, they can be scheduled serially by using control instructions such as barriers. However, naive serialization is harmful. Quantum states are extremely fragile — the quality of quantum information loses "coherence" exponentially with increasing compute time. Because of this exponential decay, current hardware and compilers such IBM Qiskit [1], Rigetti Quilc [57], and TriQ [44] schedule maximum instructions in parallel. An ideal schedule balances the need for crosstalk mitigation against the need to compute before decoherence.

Figure 2 shows our crosstalk mitigation approach. Crosstalk mitigation in software requires first, an accurate characterization of crosstalk noise present in the hardware, and second, an intelligent scheduler that uses characterization data to navigate the crosstalk-coherence tradeoff. We advance the state-of-the-art on both fronts.

Our contributions include the following. First, it is known that device characteristics affect compilation quality and program reliability[43]. However, measuring all device characteristics (akin to measuring the full process map) is an intractable problem due to exponential scaling. Therefore, the performance of such devices is typically judged based on a few metrics, such as the gate error rates and the qubit lifetimes, which are collected daily for current QC systems. This paper quantifies the degree to which crosstalk has an important effect on program reliability.

Second, since measuring crosstalk noise on every pair of simultaneous operations is computationally expensive, (requiring more than 8 hours of machine compute time even for a 20-qubit device), we develop approaches to reduce this overhead. We implement our methods in IBM Qiskit Ignis [24], an open source toolbox for device characterization.

On three 20-qubit IBM devices, our optimizations reduce characterization time to under 15 minutes.

Third, our evaluation offers insights about crosstalk noise: crosstalk can degrade the error rate of a two-qubit operation up to 11x. The degradation is not static; the effect of crosstalk on a particular gate varies up to 3x over many days. On all the three devices in our study, crosstalk noise primarily affects only nearest-neighbor gates.

Fourth, we develop an instruction scheduler that mitigates the application impact of crosstalk. We model the gate scheduling problem as a Satisfiability Modulo Theory (SMT) optimization and find optimal schedules. We implement our scheduler in IBM Qiskit Terra [21], an open-source QC compiler. Using real-system runs on three IBMQ systems, we show that crosstalk mitigation improves the error rate of SWAP circuits by up to 5.6x, geomean 2x over the parallel instruction scheduler previously used by default in IBM systems. Since SWAP operations are the fundamental method of communication in these systems, this large improvement impacts all programs that rely on communication [27, 43, 44, 55, 56, 58–60], especially as systems scale up. Our scheduler also improves the loss in cross entropy for QAOA circuits by up to 3.6x compared to the IBM scheduler. In addition, using executions on crosstalk-free regions of the hardware, we empirically verify that the mitigation provided by our approach is near-optimal in practice.

Finally, this work makes the case for software mitigation of crosstalk. This is timely as the trend in quantum computer architecture is moving towards combating crosstalk solely in hardware, either by building more sparsely-connected qubits (such as IBM systems [8]) and/or by disabling simultaneous nearby gates entirely at the hardware level (such as in Rigetti and Google's Bristlecone system [5, 6]). Both approaches impose a performance burden when mapping applications to the hardware. Instead, we argue that compilers can better navigate the design tradeoffs.

## 2 QC Background

### 2.1 Principles of Quantum Computing

A qubit is the fundamental building block of a QC system. Qubits have two basis states $|0\rangle$ and $|1\rangle$. Unlike classical bits, qubits can also be in superposition, where the state is $\alpha\,|0\rangle + \beta\,|1\rangle$, for $\alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1$. When all $n$ qubits in a QC system are in the maximal superposition state ($|\alpha|^2 = |\beta|^2 = 0.5$), the system represents $2^n$ basis states simultaneously, unlike classical systems (non-quantum) which can be in exactly one of the $2^n$ states at any given time.

Instructions or operations in a QC system are termed gates. Gates manipulate information by modifying the complex amplitudes associated with the qubit basis states. The hardware to implement QC gates is designed to apply some dynamic physical interaction to the qubit using a time-dependent set of control signals. For example, in IBMQ systems, gates are implemented by driving the qubits with microwave voltage pulses [40]. Two-qubit Controlled NOT gates are implemented using the cross-resonance effect [11, 51] where a pulse is applied on control qubit at the resonant frequency of a target qubit. This gate produces entanglement among qubits, which results in non-classical correlated behaviour.

In a QC application, an algorithm is mapped to gates which execute on a set of appropriately initialized qubits. During execution, qubit states are manipulated and the state space is evolved towards the desired output. At the end of the algorithm, a classical output bitstring can be generated using readout operations which collapse each qubit state's to $|0\rangle$ or $|1\rangle$.

### 2.2 Operational Noise in NISQ Systems

QC systems have spatial and temporal noise variations arising from manufacturing imperfections, imperfections of gate implementation and control, and external interference [30, 44]. These systems are calibrated frequently to reduce operation noise; during calibrations, error rates are measured using randomized benchmarking [34] and reported for each gate [23].

For the systems used in our study, the error rates for single qubit operations are less than 0.1%. Error rates for two-qubit CNOT gates range from 0.5-6.5%, average 1.8%. Readout error on a single qubit is 4.8% on average. These error rates indicate the reliability of the operation when it is performed in isolation. QC executions consist of sequences of gates followed by readout, and the errors compound.

While such standalone gate error rates are measured daily, error rates for the simultaneous execution of multiple of these gates have been time-consuming to characterize and therefore are not measured daily. This paper demonstrates that such simultaneous error characterizations are useful, since they can be exploited in the compiler to mitigate the impact of crosstalk.

## 3 Related Work and Novelty

A vast body of prior work exists on quantum circuit optimization to reduce the total number of gates or number of layers in the dependency graph (depth). Refs. [2, 15, 37, 45] optimize abstract program IR, without considering hardware constraints, while [9, 38, 41, 55, 59, 61] develop optimizations for mapping programs to hardware qubits to reduce the circuit size or depth. Refs. [19, 36] use commutation rules to minimize program duration. Ref [6] considers the case where gates in proximity are disabled from operating simultaneously due to crosstalk, but takes that as disabled in hardware.

Almost all prior work takes it for granted that lower program duration (a.k.a quantum circuit depth) is better, and do not consider crosstalk effects. This is intuitive, since qubits lose their information at an exponential rate as time passes.

However, in this work, we show that program duration can be traded off to avoid crosstalk, and thus improve the overall reliability of application executions.

Recently, [43, 44, 47, 58] used hardware characterization data to improve the quality of compilation. They improve the quality of mapping and SWAP insertion using independent gate error rate data measured and published by QC vendors, which does not include crosstalk characterization. Consequently, neither these works nor industrial compilers such as IBM Qiskit [1], Rigetti Quilc [57] or Google Cirq [18] consider crosstalk effects.

On the hardware side, sparse qubit connectivity [8], frequency allocation techniques [7] and gate implementation approaches [53] have been used to reduce crosstalk. These approaches are complementary to our work and are implemented in the hardware used for our evaluation. Hardware scheduling techniques have also been developed. In IBM systems, the default hardware scheduler allows maximum parallelism and aligns all gates to the right to execute them late as possible. Figure 1c shows an example. While this optimization reduces decoherence errors, it does not reduce crosstalk. In Rigetti and Google Bristlecone systems, the hardware scheduler disables simultaneous nearby gates entirely to avoid crosstalk [5, 6], irrespective of other hardware or application characteristics. This approach incurs high decoherence error because of excessive serialization. Our work proposes the first software technique for crosstalk mitigation and develops an instruction scheduler that serializes instructions to avoid crosstalk, but also balances the need to mitigate decoherence errors.

To the best of our knowledge, this work is the first to evaluate schedule qualities on real quantum systems (with real-world noise characteristics), and the first to improve schedule qualities by considering spatial as well as temporal aspects of the schedule — that is, which operations should be scheduled when and in proximity to which other operations. Our work is also the first to quantitatively show the extent to which crosstalk effects influence the reliability of programs.

# 4 Crosstalk Mitigation in Software: Design Questions

## 4.1 Background on Crosstalk Sources in Superconducting Systems

In superconducting systems, crosstalk can occur for several reasons. One type of crosstalk is due to the hardware necessary to couple pairs of qubits for two-qubit operations. There is a tradeoff between the strength of these couplings and the known, but unwanted, crosstalk they generate. In IBM devices, each qubit is connected to a few other nearest-neighbor qubits through fixed-frequency microwave resonators resulting in an always-on coupling. In Figure 3, each CNOT gate (edges) corresponds to one resonator. Because of the always-on nature of the coupling, when a control pulse is driven on one of the qubits, the resonator can propagate an unwanted drive to neighboring qubits and corrupt their state. This effect is particularly acute for nearby qubits that have similar resonant frequencies. If multiple nearest neighbor or next nearest neighbor qubits have overlapping resonant frequencies, driving a qubit can lead to unwanted state changes on other qubits. Despite meticulous efforts to mitigate crosstalk in QC hardware [8, 53], crosstalk noise is present in real devices [20, 52].

## 4.2 Characterizing Crosstalk Noise Through Randomized Benchmarking

To mitigate crosstalk noise in software, we must first characterize the hardware. For example, in Figure 1, to quantify the impact of crosstalk for the gates $g_1$ and $g_2$ executing in parallel, we have to measure the crosstalk noise for the corresponding hardware gates CNOT 0,1 and CNOT 2,3. To accomplish this, the error rate of CNOT 0,1 and CNOT 2,3 can be measured independently, without invoking any other gate. Then, the error rate of CNOT 0,1 and CNOT 2,3 can be measured simultaneously, by invoking them in parallel. If the simultaneous error rates are much higher than the independent error rates, crosstalk exists between the two gates.

Such measurements can be performed using Randomized Benchmarking (RB), a standard procedure for measuring gate error rates, which is used in IBM systems [29, 33, 34]. To measure the error rate, a single invocation of a gate is not enough. For CNOT error measurement, RB uses multiple random circuits, each having multiple invocations of the CNOT composed along with random single qubit operations. By executing these circuits on the hardware and fitting the results to a theoretical model, the error rate is estimated. For a gate $g_i$, we denote the error rate measured without invoking any other gate in the system as the *independent error rate* $E(g_i)$ and the error rate of $g_i$ measured simultaneously with $g_j$ as a *conditional error rate* $E(g_i|g_j)$. Simultaneous RB (SRB) on a pair of gates $g_i$ and $g_j$ yields both $E(g_i|g_j)$ and $E(g_j|g_i)$. When a gate $g_i$ has crosstalk interference with $g_j$, we expect $E(g_i|g_j)$ to be higher than $E(g_i)$.

While independent error rates are available from daily calibration data, conditional errors are not. To measure conditional error rates for a device, we have to perform SRB experiments between every pair of CNOT gates that can be driven in parallel i.e., CNOT pairs such as CNOT 0,1 and CNOT 2,3 that do not share a qubit. For IBMQ Poughkeepsie this approach requires 221 pairs of SRB experiments. Each such SRB experiment requires multiple runs with different random gate lengths (to get the final curve fit to the theoretical model) and each data point on the curve requires multiple trials because of noisy operations. With 100 random sequences per SRB, and 1024 trials per sequence, this baseline method requires 22.6M executions and over 8 hours of computation at current execution rates. Since QC systems have highly variable noise properties [44], daily crosstalk measurement (similar to daily gate error measurement which is already performed by IBM) will consume over a third of a device's total lifetime.

Therefore, we ask: *How can we perform crosstalk characterization experiments efficiently across the full device? Can we exploit the physical properties of the device to reduce the number of experiments? What crosstalk measurements are useful for mitigation in software?*

### 4.3 Mitigating Crosstalk by Instruction Scheduling

To avoid crosstalk, a compiler can choose to serialize the interfering operations. However, serialization can lead to decoherence errors. On IBM systems, coherence times on individual qubits range from 10-100 microseconds [44] — when a program executes for 50 microseconds on the best qubit with 100us coherence, it is 60% likely that the state is corrupted. To mitigate this dramatic loss in reliability from decoherence, the compiler should parallelize instructions as much as possible.

We ask: *How can a compiler optimize the two conflicting objectives of serializing instructions to avoid high crosstalk and parallelizing instructions to avoid decoherence? How much does crosstalk-adaptivity matter for parallel operations on superconducting devices?*

## 5 Reducing the Crosstalk Characterization Overhead

We first present detailed characterization results on three IBM systems. Using insights from our characterization, we propose optimizations to reduce the characterization overhead.

### 5.1 Characterization Results on IBMQ Systems

Figure 3 illustrates the crosstalk measurements for the three systems. Across the gate pairs, crosstalk noise increases the gate error rates up to 11x. For IBMQ Poughkeepsie, high crosstalk errors occur only in 5 gate pairs, a small fraction of the overall number of gate pairs (221). In each interfering pair, the two gates are separated by 1 hop i.e, the shortest path from one gate to the other is of length 1, which is the expected behavior from device design.

Figure 4 shows daily variations of the error rates for IBMQ Poughkeepsie. Conditional error rates for a gate vary up to 2x for IBMQ Poughkeepsie, and up to 3x on the other two systems (not shown). Even though the absolute error rates vary, the set of high crosstalk gate pairs tends to remain the same across days.

### 5.2 Our Optimizations

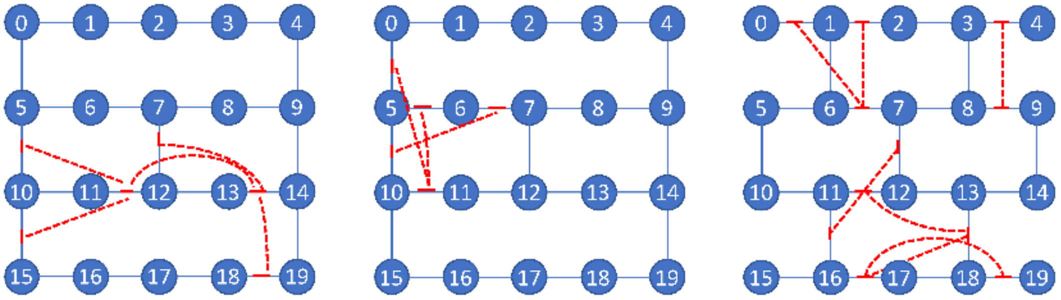
To mitigate crosstalk through instruction scheduling, we require accurate characterization data. Since crosstalk noise has spatio-temporal variations, it should be characterized daily to supply correct inputs to the compiler, similar to how gate errors and coherence times are measured daily on IBM systems. Towards this, we wish to reduce the number of experiments required to measure conditional gate error rates.

In the previous section, we measured conditional error rates for *every pair* of CNOT gates that can be driven in parallel. For IBMQ Poughkeepsie this approach requires 221 pairs of SRB experiments and over 8 hours of real-system compute time on the QC device. *All these experiments are performed on the hardware, not in simulation.* At face value, and without knowledge of the spatio-temporal behavior of crosstalk, this means that to enable compiler-level mitigation of crosstalk we must run this expensive characterization step daily. However, through a series of optimizations, we can reduce the characterization overhead.

***Optimization 1: Characterize only 1-hop pairs.*** It is sufficient to perform SRB experiments on gate pairs which are separated by 1 hop since on our devices, crosstalk noise from a gate is significant only at 1 hop distance (see Figure 3). This is the expected behavior from device design, since qubits are dispersively coupled, i.e., the ratio of coupling strength to detuning is much less than one. For each additional hop the effective coupling is suppressed by this dispersive factor. However, device packaging imperfections have been seen to introduce longer range crosstalk effects in some older systems [20, 46, 50].

***Optimization 2: Parallelize SRB experiments of multiple gate pairs.*** Next, given the above observation about lack of long-range crosstalk, we can efficiently parallelize crosstalk measurements across several gate pairs. When two pairs are separated by two or more hops, their SRB measurements can be performed in parallel. For example, in IBMQ Poughkeepsie, we can perform crosstalk measurement for the pairs (CNOT 0,1 | CNOT 2,3), (CNOT 6,7 | CNOT 8,9) and (CNOT 16,17 | CNOT 18,19) in the same experiment since each pair is at least 2 hops away from any other pair.

To efficiently parallelize SRB experiments, we can model the problem as an instance of bin packing. Given a set of $n$ gate pairs on which SRB measurements are required, we use a randomized first fit heuristic to pack the pairs into a small number of experiments. The heuristic iteratively builds a set of bins, with each bin corresponding to an experiment. Initially, there is only one empty bin. The heuristic iterates through the gate pairs and places each pair in the first *compatible* bin. A pair $(g_i, g_j)$ is compatible with a bin if all pairs $(g_k, g_l)$ in the bin are at least $k$ hops away. For example in IBMQ Poughkeepsie, with k=2, the pair (CNOT 16,17 | CNOT 18,19) is compatible with a bin which contains the pair (CNOT 6,7 | CNOT 8,9); it is not compatible with a bin which contains the pair (CNOT 11,12 | CNOT 13,14). When no existing bin is compatible, a new bin is created. All gate pairs are partitioned into a set of bins in this manner.



(a) IBMQ Poughkeepsie (b) IBMQ Johannesburg (c) IBMQ Boeblingen

**Figure 3.** Crosstalk measurement results for the three systems. The nodes are indexed by qubit id. The edges indicate two-qubit CNOT gates. Note that the number of connections is less than a regular 2D grid. For all CNOT gates, the independent gate error rate is at most 5%. We performed simultaneous RB experiments on all pairs of CNOT operations in the hardware, one pair at a time. SRB experiment on a pair $g_i$ and $g_j$ gives $E(g_i|g_j)$ and $E(g_j|g_i)$. We illustrate the data by drawing red dashed edges to indicate *high crosstalk* gate pairs i.e., all gate pairs $g_i$, $g_j$ for which the conditional error rate is much higher than the independent error rate. For this plot we selected CNOT pairs such that $E(g_i|g_j) > 3 * E(g_i)$ or $E(g_j|g_i) > 3 * E(g_j)$. For example, on IBMQ Poughkeepsie, CNOT 10, 15 has an independent error rate of 1% and conditional error rate of 11% with CNOT 11, 12.

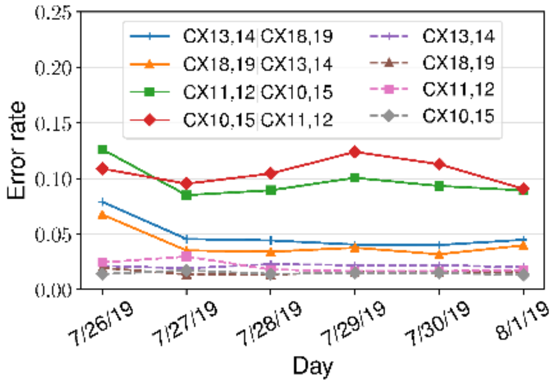


**Figure 4.** Daily variations of crosstalk noise in IBMQ Poughkeepsie. Lower error rate is better. Conditional error rates of a gate, say $E(CX13, 14|CX18, 19)$ are much higher than the independent error rate throughout the experiment week. The conditional error rates vary up to 2x on this machine, and up to 3x across devices.

We repeat the algorithm multiple times by shuffling the list of gate pairs randomly and select the partitioning with the minimum number of bins. We perform SRB experiments in parallel for all gate pairs that belong to the same bin.

***Optimization 3: Characterize high crosstalk pairs only.*** Finally, from our characterization data over several days for these devices, the set of high-crosstalk pairs remains relatively stable across days (see Figure 4). This is due to the structural nature of crosstalk pairs, and compared to gate errors, it is less prone to drift or regular changes. Hence, we can optimistically restrict our daily measurements on these pairs, and periodically, say once every few days, characterize the remaining 1 hop pairs.

Combining all optimizations, we can reduce the characterization time for the three systems to under fifteen minutes. After characterization, the data can be used by all compilation jobs in this period to improve their output.

## 6 Crosstalk Mitigation Through Instruction Scheduling: Overview

The input to our scheduler is a hardware-compliant program IR i.e., the program qubits are mapped to the hardware qubits and the IR includes the necessary SWAP instructions required to respect connectivity constraints. Figure 1b illustrates such a program IR for the example machine in Figure 1a. In our implementation using IBM Qiskit, we obtain such IR by invoking existing passes for mapping and SWAP insertion. The scheduler uses crosstalk characterization data along with machine calibration data (independent error rates, coherence time, gate duration) to determine a start time for each gate.

We pose the gate scheduling program as a constrained optimization problem to be solved by a Satisfiability Modulo Theory (SMT) solver [4, 14]. The optimization has variables and constraints which express program information and hardware error information. The variables in the optimization include the start time and error rate for each gate. We use gate dependency constraints to specify that the schedule should preserve program data dependencies.

To model the effect of crosstalk, we should determine the error rate of a gate based on the program schedule. When a gate does not overlap in time with other operations, its error rate is set using crosstalk-free independent error rates.

| Algorithm | Objective | Method |
|---|---|---|
| SerialSched | Mitigate crosstalk | Schedule all instructions serially |
| ParSched | Mitigate decoherence | Schedule maximum instructions in parallel. Current state-of-the-art, used in Qiskit [1], Quilc [57] and TriQ [44] |
| XtalkSched | Mitigate crosstalk and decoherence | SMT optimization with crosstalk characterization data. $\omega$: crosstalk weight factor. (Section 6, 7) |

**Table 1.** List of schedulers used in our evaluation.

When the gate overlaps with other operations, the error rate is based on conditional error rates with the overlapping operations. For each gate, we determine the set of overlapping operations based on the IR dependencies. The subsets of this set are the various gate overlap scenarios and are used to set the appropriate conditional error rates for a gate.

To model the effect of decoherence, we associate a *lifetime* variable with every qubit. The lifetime is the time elapsed between the first operation and the last operation on the qubit. We associate a decoherence error rate variable with a qubit which is computed as an exponential penalty on the lifetime, normalized by the coherence time of the qubit. Thus, when the lifetime increases, the decoherence error rate increases.

The objective function captures the tradeoff between instruction serialization for crosstalk mitigation and parallelization for decoherence mitigation. We minimize the product of gate error rates (which are influenced by crosstalk) and the qubit error rates (which are based on decoherence). When the optimizer serializes two gates which have high crosstalk, the gate error rate terms reduce and the decoherence terms increase. Similarly, when the gates are executed in parallel, the gate error terms increase and the decoherence terms reduce. Minimizing the objective over the entire program allows us to find the optimal schedule which mitigates crosstalk while also balancing the errors from decoherence.

Finally, to implement the schedule and enforce gate orderings, we use a post-processing step to insert control instructions in the form of barriers. We call our scheduler XtalkSched. We compare its performance to two baselines SerialSched and ParSched. These variants are discussed in Table 1.

## 7 Instruction Scheduling: Optimization Details

### 7.1 Variables

Let $Q$ be the set of qubits and $G$ be the set of gates in the IR. For each gate $g \in G$, the start time is denoted by $(g.\tau)$, duration by $(g.\delta)$, and error rate by $(g.\epsilon)$. To denote data dependencies between two operations, we use a binary relation $>$ on the gates. For two operations $g_j > g_i$ if $g_j$ depends on $g_i$. In addition to these variables, for each qubit $q$ in the program, we create a coherence error rate variable $q.\epsilon$.

### 7.2 Constraints

**Data dependency constraints:** If two gates $g_i$ and $g_j$ operate on the same qubit, and $g_j$ uses the output of $g_i$, $g_j$ should start only after $g_i$ finishes. Such dependencies can be enforced by the following constraint.

$$\forall g_i, g_j \in G : g_j > g_i \Rightarrow g_j.\tau \geq g_i.\tau + g_i.\delta \tag{1}$$

For example, for Figure 1b, the constraint $g_1 > g_0.\tau + g_0.\delta$ expresses the data dependency between $g_0$ and $g_1$.

Gate duration information is available to the compiler either from machine documentation or from calibration data and is used to set the duration variables $\delta$.

**Gate error constraints:** These constraints set crosstalk dependent error rates for each two-qubit gate. We don't consider conditional error rates based on single qubit gates because their error rates are 10x better than CNOT error rates on current systems [44]. For each gate $g_i$ denote by $CanOlp(g_i)$, the set of all operations that can overlap with it. This set can be computed by finding each $g_j$ that is neither an ancestor nor a descendent of $g_i$ in the program dependency graph specified by the IR. In Figure 1b, $CanOlp(g_2) = \{g_1, g_3\}$. $g_0$ is not considered because it is a single-qubit gate. We prune this set further to only include gates which have high conditional error rates, which in our systems are at 1 hop distance from $g_i$.

For each gate $g_j \in CanOlp(g_i)$, we create an overlap indicator $o_{ij}$, which tracks whether $g_i$ and $g_j$ overlap in the schedule. $o_{ij}$ is set using the following constraint.

$$o_{ij} = (g_j.\tau \leq g_i.\tau + g_i.\delta \wedge g_i.\tau \leq g_j.\tau + g_j.\delta) \tag{2}$$

How can we set the gate error rates using the overlap indicators? Consider $g_2$ in Figure 1b. Since $g_2$ can overlap with $g_1$ and $g_3$, there are 4 possible scenarios: both $g_1$ and $g_3$ don't overlap with $g_2$, only $g_1$ overlaps with $g_2$, only $g_3$ overlaps with $g_2$, and both $g_1$ and $g_3$ overlap with $g_2$. For each case, we set error rates using the following constraints.

$$\neg o_{12} \wedge \neg o_{13} \Rightarrow g_2.\epsilon = E(g_2) \tag{3}$$

$$o_{12} \wedge \neg o_{13} \Rightarrow g_2.\epsilon = E(g_2|g_1) \tag{4}$$

$$\neg o_{12} \wedge o_{13} \Rightarrow g_2.\epsilon = E(g_2|g_3) \tag{5}$$

$$o_{12} \wedge o_{13} \Rightarrow g_2.\epsilon = max\{E(g_2|g_1), E(g_2|g_3)\} \tag{6}$$

In constraint 3, the error rate is the independent error rate of $g_2$, since it doesn't overlap with the other two gates. In constraint 4, the error rate is the conditional rate of $g_2$ with $g_1$. In constraint 6, when both gates overlap with $g_2$, crosstalk may arise from both gates. But, in order to conservatively serialize gates, we only consider crosstalk from the worst gate, and take the maximum error rate over the two overlapping gates. (We have not observed significant worsening of errors from simultaneous execution of triplets of gates).

We generalize these constraints as follows. To set the error rate for $g_i$, we enumerate all possible overlap scenarios by considering the powerset[1] of $CanOlp(g_i)$. For each nonempty subset $Olp_k$ in the powerset, we denote the complement by $NotOlp_k$ i.e., $NotOlp_k = CanOlp(g_i) \setminus Olp_k$ and we add the following constraint.

$$\bigwedge_{g_j \in Olp_k} o_{ij} \bigwedge_{g_j \in NotOlp_k} \neg o_{ij} \Rightarrow g_i.\epsilon = \max_{g_j \in Olp_k} E(g_i|g_j) \tag{7}$$

In other words, when the gates in the set $Olp_k$ overlap with $g_i$, and the gates in the set $NotOlp_k$ don't overlap with $g_i$, the constraint sets the error rate to be the maximum conditional error rate over the overlapping gates.

For the empty subset in the powerset, we add the following constraint to account for the case when none of the gates overlap with $g_i$.

$$\bigwedge_{g_j \in CanOlp(g_i)} \neg o_{ij} \Rightarrow g_i.\epsilon = E(g_i) \tag{8}$$

Although there are $2^{|CanOlp(g_i)|}$ constraints for each gate, in practice the size of the set will not be large because it includes only overlapping gates with high conditional error rates. As Figure 3 shows, this is small for our systems.

**Decoherence error constraints:** These constraints track the decoherence errors on each qubit in the program. They use coherence time measurements available from daily machine calibration.

Exponential state decay in qubits can occur in two ways: $T_1$ time for the state $|1\rangle$ to decay to $|0\rangle$ and $T_2$ time for a superposition state $(|0\rangle + |1\rangle)/\sqrt{2}$ to decay to either $|0\rangle$ or $|1\rangle$. These are relaxation and dephasing respectively. We use the term decoherence to refer to both these effects. $T_1$ and $T_2$ values are reported for each hardware qubit during daily calibration. If a program performs computation for time $t$ on a qubit, the probability of error from $T_1$ losses is proportional to $1 - e^{-t/T_1}$, and the probability of error from $T_2$ losses is proportional to $1 - e^{-t/T_2}$. When $t$ increases, the error rate increases exponentially.

We set the decoherence error rate for a qubit $q_i \in Q$ by computing the *lifetime* of the qubit in the schedule. The lifetime $q_i.t$ is the difference between the finish time of $q_i$'s last gate $L(q_i)$ and the start time of $q_i$'s first gate $F(q_i)$. Current QC systems are typically limited by $T_1$ errors, but on some qubits, $T_2$ times can be much lower than $T_1$ because of noise fluctuations. To consider such cases, we set the maximum available compute time $q_i.T$ as the minimum of $T_1$ and $T_2$ values of the qubit. We set the decoherence error on a qubit as follows.

$$q_i.t = L(q_i).\tau + L(q_i).\delta - F(q_i).\tau \tag{9}$$

$$q.\epsilon = 1 - e^{q_i.t/q_i.T} \tag{10}$$

Although this constraint performs exponentiation over an optimization variable, in the next section we show that it can be expressed as a linear term.

**IBMQ-specific constraints:** Using Qiskit at the circuit level, we can enforce control dependencies only using barrier instructions. Therefore, any schedule where two gates partially overlap cannot be enforced using the circuit-level ISA [13][2]. For each gate $g_i$, and for $g_j \in CanOlp(g_i)$ we enforce that the two gates can either be scheduled without any overlap or such that one of them happens fully within the duration of the other.

$$(g_i.\tau + g_i.\delta < g_j.\tau) \lor (g_j.\tau + g_j.\delta < g_j.\tau) \lor \tag{11}$$

$$((g_i.\tau + g_i.\delta < g_j.\tau + g_j.\delta) \land (g_i.\tau > g_j.\tau)) \lor \tag{12}$$

$$((g_j.\tau + g_j.\delta < g_i.\tau + g_i.\delta) \land (g_j.\tau > g_i.\tau)) \tag{13}$$

In current IBMQ systems the hardware control forces all readout operations to occur simulateneously at the end. Therefore, all gates are right-justified and scheduled from the end. This affects the qubit lifetime variables in our optimization. We model this behavior with a constraint that equates the start times of all readout operations in the program.

### 7.3 Objective Function

Ideally, to minimize both gate errors from crosstalk and decoherence errors we can set the objective as,

$$\min \Bigg( \underbrace{\prod_{\forall g \in G} (g.\epsilon)}_{\text{Gate errors (crosstalk)}} \underbrace{\prod_{\forall q \in Q} (q.\epsilon)}_{\text{Decoherence error}} \Bigg). \tag{14}$$

The first term minimizes the product of the gate errors and the second term minimizes the product of decoherence errors. Since the SMT solver requires linear operations, we can minimize the logarithm of the objective to get a linear function.

$$\min \Bigg( \sum_{\forall g \in G} (\log g.\epsilon) + \sum_{\forall q \in Q} (\log q.\epsilon) \Bigg) \tag{15}$$

By substituting the definition for $q.\epsilon$ from constraint 10, we can re-write the objective as follows.

$$\min \Bigg( \sum_{\forall g \in G} (\log g.\epsilon) - \sum_{\forall q \in Q} (q.t/q.T) \Bigg) \tag{16}$$

In this form the objective function clearly shows the crosstalk-coherence tradeoff. When gates are serialized to reduce crosstalk errors, the first term reduces and the second term increases, and vice versa when gates are parallelized.

[1] The powerset of a set S is the set of all subsets of S, including the empty set and S itself. The cardinality of the powerset is $2^{|S|}$.

[2] Recent versions of Qiskit and IBMQ systems provide a pulse-level abstraction for more fine-grained control of systems [39]

Finally, to test the relative importance of crosstalk and decoherence errors, we consider a weighted objective where a crosstalk weight factor $\omega \in [0, 1]$ is applied to the gate error rate terms.

$$\min\left(\omega \sum_{\forall g \in G} (\log g.\epsilon) - (1-\omega) \sum_{\forall q \in Q} (q.t/q.T)\right) \tag{17}$$

To compute the optimal schedule for a program, we first use Qiskit's passes to generate the program IR and map it to the hardware. The mapped program IR is used to create the optimization problem using this objective along with data dependency, gate error and decoherence error constraints. These constraints make the $g.\epsilon$ and qubit lifetime $q.t$ variables dependent on the gate schedule. The gate schedule produced by the optimization is post-processed to generate executable code with the barriers necessary to enforce the optimal gate orderings. We call this algorithm XtalkSched.

## 8 Experimental Setup

### 8.1 Crosstalk Characterization Implementation

We implemented the crosstalk characterization methods using IBM Qiskit Ignis version 0.2.0 [24], an open-source framework for error characterization. For CNOT error characterization, RB applies random sequences of *m* two-qubit Clifford gates which are constructed from single qubit gates and multiple invocations of the CNOT. The final gate is the inverse of the previous gates so that the sequence should return to the original state. By measuring the final state as a function of *m* and fitting to a theoretical model, one can extract the error rate per Clifford. To extract the CNOT error rate, the Clifford error rate is divided by the by the number of CNOTs per Clifford (optimally 1.5). This assumes the single qubit gate error is negligible and gives an upper bound on CNOT error. To measure a CNOT gate's error rate independently, we apply standard two-qubit RB [24]. To measure the error rates for two CNOTs simultaneously, we apply simultaneous two-qubit RB (SRB) [16]. In each SRB experiment, we used 100 random sequences, with up to 40 Clifford gates per sequence and performed 1024 trials per sequence.

### 8.2 Instruction Scheduler Implementation and Baselines

We implement our instruction scheduler XtalkSched as a compilation passes in IBM Qiskit Terra version 0.8.2 [21], an open-source compiler framework. The SMT optimization for XtalkSched uses the Z3 SMT solver [14] version 4.8.4, using the Z3py APIs. We test our scheduler in comparison to two baselines SerialSched and ParSched, shown in Table 1. SerialSched serializes all operations in the program. ParSched is the current state-of-the-art scheduler used in IBM Qiskit.

### 8.3 Benchmarks

**SWAP Circuits:** We demonstrate the importance of crosstalk-adaptivity for communication orchestration in superconducting QC systems which have nearest-neighbor connectivity. In these architectures CNOTs are permitted only between adjacent qubits. To enable a CNOT between two far away qubits, compilers insert a sequence of SWAP operations which move the qubits into adjacent locations through exchanges. For example, in IBMQ Poughkeepsie, `CNOT 0,13` can be implemented as `SWAP 0,5; SWAP 5,10; SWAP 13,12; SWAP 12,11; CNOT 10,11;`, where both qubits meet-in-the-middle. Each SWAP operation is in turn composed of three CNOT gates[3]. Figure 6b shows the operations executed for this sequence.

We create meet-in-the-middle SWAP sequences between pairs of qubits in the device and schedule it using the three algorithms. When SWAP paths are executed on qubits which have no crosstalk e.g., on the path 0, 1, 2, 3 on IBMQ Poughkeepsie (see Figure 3), XtalkSched and ParSched produce the same schedule. We avoid such SWAP paths in our evaluation and focus on 46 circuits across the three devices which include at least one pair of high crosstalk CNOTs.

**QAOA Circuits:** We ran experiments on QAOA, a promising NISQ application, using the hardware efficient ansatz [42]. We used circuits with 4 qubits and 43 gates (9 two-qubit gates). We performed experiments on four crosstalk-prone regions in IBMQ Poughkeepsie.

**Other benchmarks:** We also study our algorithm on the Hidden Shift benchmark [10] used in prior work [43, 44]. We use Hidden Shift instances for sensitivity studies. Similar to SWAP circuits, we create instances of these circuits on subsets of qubits which are affected by crosstalk. To test scaling, we use instances of quantum supremacy circuits [35].

### 8.4 Metrics

**Crosstalk characterization:** We count the number of SRB experiments and time required to perform characterization. We compare these metrics for a policy which performs SRB experiment all pairs of gates in the device, and with the three optimizations proposed in Section 5.

**Instruction scheduling:** For SWAP circuits, we setup the circuit such that it creates a known answer, that can be measured (a Bell state) which can be measured using state tomography [24]. State tomography provides an error rate in the range [0, 1], with 1 meaning that the state is created perfectly. To execute state tomography we use 9216 trials (1024 per basis pair * 9 basis pairs) on the real system to obtain the error rate.

For QAOA circuits, the output is obtained using 8192 trials. Since the output is a distribution of states, we used cross-entropy to measure the similarity of the output to the ideal

[3] `SWAP 0,1 := CNOT 0,1; CNOT 1,0; CNOT 0,1`

theoretical distribution. For Hidden Shift, we perform 8192 trials. The expected output is a single bit string, therefore, the error rate is measured as fraction of trials which did not yield the correct bit string.

In all cases, readout error mitigation [25] is used to reduce the effect of imperfect hardware readout operations.

### 8.5 Setup

Our compilation experiments use an Intel Core i7 processor (2.6GHz, 32GB RAM) with Python 3.7. We use three 20-qubit IBM systems for the quantum experiments. The device interface APIs in IBM Qiskit [1] were used to run characterization and application circuits. The daily machine calibration data is available through these APIs. The calibration data includes gate durations and independent error rates for all gates and coherence time ($T_1$ and $T_2$) for all qubits.

## 9 Optimizing Application Error Rate

### 9.1 Comparisons to Baselines using SWAP Circuits

**Improvement in Error Rate:** Figure 5 compares the error rate for SWAP circuits scheduled with XtalkSched $\omega = 0.5$, versus the SerialSched and ParSched schedulers, on the three systems. Although SerialSched naively serializes all instructions, in some cases it offers lower error than ParSched, because it avoids high crosstalk. ParSched outperforms SerialSched because it avoids decoherence by parallelizing operations. On all the tested qubit pairs, XtalkSched has significantly lower error rate than SerialSched and ParSched because it optimizes both crosstalk and decoherence. On IBMQ Poughkeepsie, XtalkSched obtains up to 4.9x reduction in error compared to ParSched, and up to 9.2x reduction compared to SerialSched. Across systems, the maximum improvement over ParSched is 5.6x, geomean 2x.

**Impact on Program Duration:** We compare the durations of schedules produced by the three algorithms. Figure 5d shows the program durations for SWAP circuits on IBMQ Poughkeepsie. Across different qubit pairs, SerialSched has the highest duration and ParSched has the lowest duration. XtalkSched produces executables which are only 1.16x longer than ParSched on average, worst case 1.7x. For NISQ applications, the most important figure of merit is the likelihood of correct execution, and not execution time. Nevertheless, XtalkSched needs to expend only a small increase in the execution time to mitigate crosstalk.

**Example Case:** Figure 6 shows the schedules for the swap path between qubit 0 and 13 on IBMQ Poughkeepsie. SerialSched schedules all 4 SWAPs in series and avoids crosstalk errors. But, it has high schedule length and therefore, high decoherence error. ParSched schedules the two pairs of logically independent SWAPs in parallel which reduces the execution time and decoherence errors. But, it incurs high crosstalk errors for the SWAP operation on qubits 5, 10 and the SWAP on 11, 12. XtalkSched obtains the best of both cases. It parallelizes the far away SWAPs which don't have crosstalk, and serializes the nearby SWAPs. This allows it to avoid the crosstalk noise, which compensates for a small increase in decoherence and improves the overall error rate.

For serializing the two SWAPs XtalkSched chooses the best ordering of operations i.e., when two gates $g_i$ and $g_j$ need to be serialized, it decides whether $g_i$ should be placed before or after $g_j$. For this system, qubit 10 has very low coherence time (less than 6us, which is nearly 10X lower than the average coherence on this system). On the IBM systems, decoherence effects on a qubit start only after the first gate is applied. If XtalkSched performs SWAP 5,10 first, followed by SWAP 11,12, the state of qubit 10 would decohere. Instead, since we model qubit lifetime to start at the first gate (constraint 9), XtalkSched computes an optimal ordering where SWAP 5,10 gets placed with minimum lifetime, after SWAP 11,12.

**Optimality:** For qubits affected by crosstalk on IBMQ Poughkeepsie, Figure 7 compares XtalkSched swap error rates to the ideal crosstalk-free error rates. To obtain the ideal error rates, we averaged swap error rates on crosstalk-free swap paths in IBMQ Poughkeepsie, selecting the lowest error schedule for each path. Figure 7 shows that XtalkSched error rates are close to the ideal and within geomean $1\% \pm 16\%$ (1 standard deviation) of average error rate of crosstalk-free swap paths of the same length.

*Given fundamental connectivity restrictions on superconducting QC systems, SWAP-based communication is important for all programs run on these systems, especially as devices and programs scale up. XtalkSched's near-optimal crosstalk mitigation and improved error rate is therefore very relevant for reliable execution on current and near-term NISQ systems.*

### 9.2 Evaluation on QAOA Circuits

Figure 8 shows the cross entropy for QAOA circuits on IBMQ Poughkeepsie using XtalkSched with $\omega \in [0, 1]$. Cross entropy measures how close the output distribution is to the ideal distribution obtained from a noise-free simulation with Qiskit Aer simulator. With $\omega = 1$ XtalkSched considers only crosstalk noise and ignores decoherence. Hence it serializes all instructions similar to SerialSched. With $\omega = 0$, only decoherence is considered, and XtalkSched is equivalent to ParSched. When $\omega$ is varied from 0.03 to 0.2, XtalkSched outperforms both the baselines and significantly reduces the cross entropy. XtalkSched reduces the loss in cross-entropy (with respect to the ideal) by geomean 1.8x (up to 3.6x) compared to ParSched and geomean 2x (up to 4.3x) compared to SeriesSched. Further, we performed experiments on crosstalk-free regions of the hardware to measure the average cross entropy achievable on the device. Owing to variability in gate errors across the device, this value has mean 1.67 and standard deviation 0.15 and is indicated by