Qiskit Pulse: Programming Quantum Computers Through the Cloud with Pulses

Thomas Alexander,[1, *] Naoki Kanazawa,[2, *] Daniel J. Egger,[3] Lauren Capelluto,[1] Christopher J. Wood,[1] Ali Javadi-Abhari,[1] and David McKay[1]

[1]*IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA*
[2]*IBM Research - Tokyo, 19-21 Nihonbashi Hakozaki-cho, Chuo-ku, Tokyo, 103-8510, Japan*
[3]*IBM Research - Zürich, Säumerstrasse 4, 8803 Rüschlikon, Switzerland*

The quantum circuit model is an abstraction that hides the underlying physical implementation of gates and measurements on a quantum computer. For precise control of real quantum hardware, the ability to execute pulse and readout-level instructions is required. To that end, we introduce Qiskit Pulse, a pulse-level programming paradigm implemented as a module within Qiskit-Terra [1]. To demonstrate the capabilities of Qiskit Pulse, we calibrate both un-echoed and echoed variants of the cross-resonance entangling gate with a pair of qubits on an IBM Quantum system accessible through the cloud. We perform Hamiltonian characterization of both single and two-pulse variants of the cross-resonance entangling gate with varying amplitudes on a cloud-based IBM Quantum system. We then transform these calibrated sequences into a high-fidelity CNOT gate by applying pre and post local-rotations to the qubits, achieving average gate fidelities of $F = 0.981$ and $F = 0.979$ for the un-echoed and echoed respectively. This is comparable to the standard backend CNOT fidelity of $F_{CX} = 0.984$. Furthermore, to illustrate how users can access their results at different levels of the readout chain, we build a custom discriminator to investigate qubit readout correlations. Qiskit Pulse allows users to explore advanced control schemes such as optimal control theory, dynamical decoupling, and error mitigation that are not available within the circuit model.

I. INTRODUCTION

In quantum computing, information is stored and processed according to the laws of quantum mechanics [2]. The primary quantum programming paradigm is the circuit model. In this model, the underlying dynamics of the physical system implementing the quantum computer are abstracted as a sequence of unitary gate operations and projective measurements applied to a set of qubits. Gates manipulate the states of qubits, while measurements extract classical information in the form of bitstrings, which encode the outcome of projective measurements of the qubits in a particular measurement basis.

Qiskit is an open-source quantum computing framework designed to enable research on near-term quantum computers and their applications. It provides tools for creating, manipulating and running quantum programs on quantum systems independent of their underlying technology and architecture. The standard programming abstraction for a quantum circuit is a quantum assembly language (QASM) such as OpenQASM [3] which Qiskit supports [1] and many similar languages that have been described in the literature [4, 5]. However, hardware is not capable of natively implementing quantum instructions and must compose these operations from the classical stimulus avilable to control hardware.

At the hardware level the time-dependent dynamics of a quantum system interacting with applied control fields is described by its Hamiltonian and the Schrödinger equation. Through careful engineering of applied classical control fields a quantum system may be steered through a desired unitary evolution [6]. Superconducting transmon qubits, for example, encode a qubit in a nonlinear oscillator formed by a parallel circuit consisting of a Josephson junction and capacitor, and may be manipulated by applying shaped microwave control pulses [7]. Implementing the quantum circuit model on such an architecture requires compiling circuit instructions to a set of microwave control instructions, or pulses, which enact the desired state-transformations and/or measurements.

In the circuit domain, an atomic circuit instruction is agnostic to its pulse-level implementation on hardware. Extracting the highest performance out of quantum hardware requires the ability to craft a pulse-level instruction schedule, which cannot be done within the standard circuit model. To enable pulse-level programming an instruction set, OpenPulse [8], was developed to describe quantum programs as a sequence of pulses, scheduled in time. We present within this paper a Python implementation of OpenPulse, *Qiskit Pulse* which adds to the Qiskit compilation pipeline the capability to schedule a quantum circuit into a pulse program intermediate representation, perform analysis and optimizations, and then compile to OpenPulse object code to execute on a quantum system.

The various hardware architectures used for current-day quantum computing systems creates a need for a pulse-level instruction set that may address most platforms at an abstract level, compatible with both commercial off-the-shelf and proprietary control instruments, including arbitrary waveform generators (AWG), signal generators, filters, amplifiers and digitizers [9]. To program such systems at the pulse-level in a *hardware independent* manner requires the user-level instruction set to be target-compiled to the underlying system hardware components, each of which may have a unique instruc-

* These two authors contributed equally;
Corresponding author: talexander@ibm.com

tion set and programming model. Recent efforts to construct microarchitectures that conform to classical computer engineering paradigms [10–12] have programming semantics closely tied to the underlying microarchitecture. Compiling directly from the circuit model to target hardware obfuscates the underlying pulses that manipulate the hardware removing a powerful degree of control.

With Qiskit Pulse we enable the development of a common and reusable suite of technology-independent quantum control techniques [6] that operate at the level of analog stimulus which may be remotely retargeted to cloud-based quantum computing systems.

Paper outline — In Sec. II we present our pulse programming model. We demonstrate the capabilities of Qiskit Pulse in Sec. III where we show Hamiltonian characterization of the two-qubit cross-resonance interaction, and calibration of a high-fidelity entangling gate, on a cloud-based quantum computer available on the IBM Quantum Experience. We discuss how the readout of quantum computers is incorporated in Qiskit Pulse in Sec. IV and conclude in Sec. V. The source code and data for the experiments within this paper has been made available online [13].

II. QISKIT PULSE PROGRAMMING MODEL

In the standard quantum circuit model, the time elapsed between operations is irrelevant as long as the order of non-commuting gates is preserved [14]. However, when controlling quantum hardware at the pulse level, properly timing and synchronizing instructions is crucial for accurately enacting quantum operations. For instance, users may create new gate definitions, characterize and correct for crosstalk on qubits neighboring interacting qubits, implement optimal control techniques such as GRAPE [15] or mitigate errors through Richardson extrapolation [16–18].

We envision that a classical microprocessor with an embedded *pulse coprocessor* will be responsible for controlling and measuring the quantum device. Within this work we only focus on describing a virtual execution model and limited set of instructions for the pulse coprocessor which can be compiled to the instruction set architecture (ISA) of the underlying control hardware. Qiskit Pulse's position in the predicted quantum computing compilation pipeline is demonstrated in Fig. 1. We expect that as quantum hardware continues to be refined these abstractions will be extended.

Qiskit Pulse provides an open source, front-end implementation of the OpenPulse interface [8]. Third parties can fully integrate with Qiskit Pulse by implementing their own Qiskit provider [1] which is responsible for translating Qiskit Pulse programs to executable programs on the provider-specific hardware which might include components such as AWGs and digitizers. Qiskit Pulse programs are composed of *pulses*, *channels*, and *instructions* which we present in the following subsections.

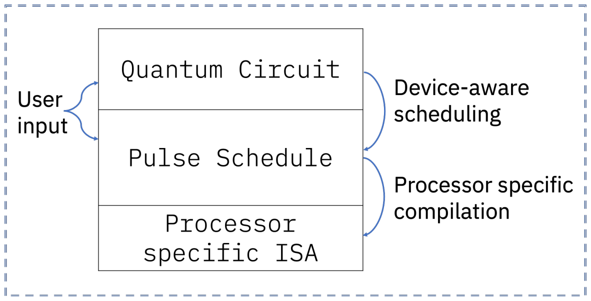

FIG. 1: The envisioned quantum program representations and their associated lowering compilation procedures. QASM programs may be built and optimized with information of the system topology, native gates, and error rates, and then are *scheduled* into pulse programs by using calibrated native gate definitions. Pulse programs are compiled to a processor-specific ISA through a target code generation procedure. The typical user is expected to program at the circuit level, whereas Qiskit Pulse enables advanced users to control at the pulse level.

A. Pulses

A *pulse* is a *time-series* of complex-valued amplitudes with a maximum unit norm, $[d_0, \ldots, d_{n-1}]$. Each d_j, $j \in \{0, ..., n-1\}$, is called a *sample*. Every system specifies a cycle-time of `dt` which is the finest time-resolution exposed on the pulse coprocessor, and is typically defined by the sample rate of the coprocessor's waveform generators. Each sample in a pulse is output for one cycle, a *timestep*. All pulse durations and timesteps are defined and discretized dimensionlessly with respect to `dt`. The ideal output signal has amplitude

$$D_j = \mathrm{Re}\left[e^{i2\pi f j\mathtt{dt}+\phi} d_j\right] \tag{1}$$

at time $j\mathtt{dt}$, where f and ϕ are a modulation frequency and a phase. The pulse samples describe only the envelope of the produced signal which are then mixed up in hardware with a carrier signal defined by a frequency and a phase. To reduce encoding sizes we also allow hardware providers to define parametric pulse shapes. For example, one parametric pulse supported by IBM backends and made available through the Pulse library is the `Gaussian` pulse. It takes three parameters: an integer `duration` in terms of `dt`, a complex amplitude `amp`, and standard deviation `sigma`. This parametric pulse can be instantiated within Qiskit in the following way:

```python
from qiskit.pulse.pulse_lib import Gaussian

duration = 128
amp = 0.2
sigma = 16
gaussian_pulse = Gaussian(duration, amp, sigma)
```

TABLE I: A summary of channels and the pulse instructions that are defined on them. Note that the `DriveChannel`, `MeasureChannel`, and `ControlChannel` are subtypes of `PulseChannel`, whereas the `AcquireChannel`, which cannot transmit stimulus pulses, is not. In the near term it is expected that the instruction set below will continue to be expanded.

Channel	Alias	Description
`PulseChannel`	-	Generic transmit channel used to manipulate the quantum system. Current supported sub-types are `DriveChannel`, `MeasureChannel` and `ControlChannel`.
`DriveChannel`	d_i	Transmit channel connected to qubit i, with signals typically modulated at a frequency in resonance with qubit i.
`MeasureChannel`	m_i	Transmit channel connected to the readout component of qubit i.
`ControlChannel`	u_i	Transmit channel with signals typically associated with arbitrary interaction terms in the Hamiltonian.
`AcquireChannel`	a_i	Receive channel connected to the readout component of qubit i, capable of digitizing and acquiring data.

Instruction	Operands	Description
`Play`	`pulse: Pulse`, `channel: PulseChannel`	Output the waveform described by `pulse` on the `channel`.
`Delay`	`duration: int`, `channel: Channel`	Idle the `channel` for the given `duration`.
`ShiftPhase`	`phase: float`, `channel: PulseChannel`	Shift the phase of the `channel` by `phase` radians.
`SetFrequency`	`frequency: float`, `channel: PulseChannel`	Set the frequency of the `channel` to `frequency` Hz.
`Acquire`	`duration: int`, `channel: AcquireChannel`, `register: Register`	Trigger the `channel` to collect data for the given `duration`, and store the measurement result in a `register`.

B. Channels

Hardware components are modeled with `Channels`. Channels label signal lines that either transmit or receive signals between the control electronics and the quantum device. Each channel executes instructions from a first-in, first-out (FIFO) queue as outlined in subsection II C.

Channels are constrained at target code generation time to target hardware components, e.g., an AWG. The calibrated parameters of a channel, such as its frequency, and the pulses played on that channel, depend on the physical properties of the targeted qubit. Therefore, channels are not interchangeable at the pulse abstraction layer, i.e. permuting channels over qubits will not give equivalent results. This highlights another difference between circuit and pulse instructions, the parameters of gates in a quantum circuit do not depend on the physical properties of the targeted qubits. The qubits in a quantum circuit can therefore be interchanged without affecting the computational result as long as the topology of the device is taken into account and gate imperfections are ignored.

There are several different channel types, and each may support a different instruction set. A summary of channels and their descriptions is provided in Table. I. The channel type, and thus the supported instructions, is determined by the effect of the channel on the quantum device. For example, a `PulseChannel` models the output of a control field $\alpha_k(t)$ in a system Hamiltonian $\hat{H}(t) = \hat{H}_{\rm sys} + \sum_k \alpha_k(t)\hat{H}_k$ which is composed of time-independent system and time-dependent control terms. The Hamiltonian term for a given channel, $\hat{H}_k$ in general may be arbitrary, but is typically associated with the subtype of the respective pulse channel that is assigned at system configuration time.

The sub-types of `PulseChannels` are `DriveChannels`, `MeasureChannels`, and `ControlChannels`. Each pulse channel maintains an instruction writeable frequency f and phase ϕ, which modify the channel output as per Eq. (1). Tracking the phase in this way enables the implementation of *virtual* Z-gates [19, 20]. Qubit drive and readout pulses are respectively assigned to `DriveChannels` and `MeasureChannels`, see e.g., the drive pulses on drive channels `d0` and `d1` in Fig. 2. Their index is trivially mapped to the address of the target qubit. The `ControlChannel` implements any remaining control fields, such as coupler drives or two-qubit drives as depicted by `u2` in Fig. 2. The backend hardware may choose to map multiple `PulseChannels` to the same control unit in the system, which enables tracking a unique phase for each channel. For example, in the IBM Quantum systems used to perform the experiments within this paper, every `DriveChannel` may share an AWG with multiple `ControlChannels`, each of which have a frequency and phase adjusted to track that of their respectively coupled qubits, enabling the implementation of two-qubit gates as demonstrated in Sec. III.

The `AcquireChannel` is used to communicate to the system when qubit readout data must be acquired. It is not associated with a control term in the Hamiltonian and does not output stimulus to the quantum system. Data collected on these channels are used to determine the qubit state, see Sec. IV for more details.

For convenience we alias drive, measurement, control and acquisition channels as $\{d_0, \dots, d_{n_q-1}\}$, $\{m_0, \dots, m_{n_q-1}\}$, $\{u_0, \dots, u_{n_u-1}\}$, and $\{a_0, \dots, a_{n_q-1}\}$ respectively, where n_q is the number of qubits and n_u is the number of arbitrary control channels of the system.

C. Instructions and Execution Model

`Instructions` may be scheduled on `Channels` to manipulate the quantum system. Pulse instructions have as operands channels and instruction-dependent constants. All pulse instructions have a fixed, deterministic duration, which may be specified either implicitly or explicitly. Instructions are executed with an *allocation* and *trigger* timing model in which instructions are loaded into a FIFO queue unique to each channel and then execution is initialized synchronously across all channels with an external trigger signal. Consequently the absolute start and end of every pulse instruction may be scheduled at compile-time across channels with hard real-time deadlines relative to the external trigger signal. Instructions may have multiple channels as operands causing an execution dependency. In this case channels stall execution until all operand channels are available. We now outline the different types of instructions, which are summarized in Table I.

Every channel supports a `Delay` instruction which has as operands a `duration` which is specified as a number of cycles, and a target channel. The channel will idle for the duration of the instruction.

The `Play` instruction allows users to play a `pulse` on a target `PulseChannel` with a frequency and phase set with the `ShiftPhase` and `SetFrequency` instructions. The `ShiftPhase` instruction has an implicit duration of zero and accepts an input float `phase` and `PulseChannel`. The `ShiftPhase` will shift the phase ϕ of the target channel by `phase` radians. This relative shift persists on the channel from the time of the instruction, allowing the phase of a channel to be accumulated throughout an experiment. The `SetFrequency` instruction has an implicit duration of zero and accepts an input float `frequency` and a `PulseChannel`. This instruction will set the frequency f of all proceeding pulses on the target channel to `frequency` Hz. Like any other instruction, `SetFrequency` can be used on a single channel multiple times within a schedule, subject to the instantaneous bandwidth of the hardware. It can therefore be used, for example, to measure the anharmonicity of a transmon qubit.

The `Acquire` instruction has as operands a `duration`, an `AcquireChannel`, and a classical `register` in which to store the observed result. This instruction signals to the measurement unit to begin acquiring data, and for how long. Each `Acquire` instruction should align with a corresponding measurement stimulus `Play` instruction to induce a measurement of the target qubit. An acquisition channel outputs an unsigned integer value `N` into the result register. For the standard two-level qubit, this will be a single bit $\{0, 1\}^1$.

If a measurement stimulus pulse measures multiple qubits, as is typical for multiplexed measurement schemes [21, 22], an acquisition instruction must be synchronously scheduled for each of the measured qubits. In hardware, each `AcquireChannel` is constrained to a measurement chain which usually includes data acquisition, filtering, kerneling, and state discrimination. To accommodate the heterogeneous readout schemes encountered in hardware we defined three levels of readout data and how to convert between them, see Sec. IV.

The set of operations in Qiskit Pulse should have sufficient generality to program a pulse coprocessor for an arbitrary quantum computing system in the time-domain and be embedded within a larger instruction set that might include both classical control flow and a traditional gate-level description of a quantum program with instructions being implemented by a lowering procedure to pulse instructions. The benefit of this approach is that a single software stack may provide the middle-end for the rapidly developing heterogeneous quantum computing platforms.

D. The Pulse Schedule

The pulse `Schedule` is the representation of a pulse program in Qiskit Pulse and is an ordered collection of scheduled pulse instructions. The pulse schedule is equivalent to a basic block [23] in a classical computation with deterministic instruction durations. To construct a pulse schedule `Instructions` may be appended as demonstrated in the example below, which prepares qubit 0 in the $|1\rangle$ state and then measures it:

```python
# Create a pulse schedule.
sched = Schedule(name='excited_state')

# Create gate and measurement pulses.
x180 = Drag(x_dur, x_amp, x_sigma, x_beta)
measure = GaussianSquare(m_dur, m_amp, m_sigma, m_square_width)

# += appends an Instruction to a Schedule.
sched += Play(x180, DriveChannel(0))

# Measure qubit 0.
sched += Play(measure, MeasureChannel(0))

# Determine the state of qubit 0 and store it
# in a persistent MemorySlot register which
# will be returned in the program result.
sched += Acquire(AcquireChannel(0), MemorySlot(0))

# Run the schedule and get the result.
counts = execute(sched, backend).result().get_counts()
```

E. Scheduling

Quantum circuits and pulse schedules are both representations of a quantum program. The Qiskit *transpiler* optimizes quantum circuits according to the properties of the targeted quantum system such as the device topology, the native gate set, and the gate fidelities. A *scheduler* compiles a circuit program to a pulse program, as depicted in Fig. 1. Scheduling requires system dependent information, most notably the definitions of the native gates in terms of scheduled pulse instructions. The scheduler therefore requires a quantum circuit to be transpiled to the native gate set of the target system prior to scheduling. Furthermore, during scheduling it is crucial to maintain the relative timing of groups of pulse instructions calibrated to implement an element of the native gate set. For instance, the cross-talk cancellation tone of a cross-resonance gate applied to the target qubit must be aligned with the pulse that drives the control qubit at the frequency of the target qubit [24].

The input circuit provides only implicit topological timing constraints, which allows the scheduler to arbitrarily resolve the remaining free time-alignment parameters in the output schedule. The scheduler's behavior in resolving free parameters is set by specifying a *scheduling method* or *policy*. By default the Qiskit scheduler follows an "as-late-as-possible" scheduling method [1]. This will schedule individual gates as late as possible while minimizing the deadtime between instructions on the same channel. This scheduling routine mitigates T_1 and T_2-decay errors by maximizing the time that qubits will spend in their initial ground state prior to the first pulse, while also minimizing the time between the last pulse and the measurement. Fig. 2 provides a code snippet for scheduling a quantum circuit into a pulse schedule using Qiskit, and visually demonstrates the correspondence between the circuit instructions input to the scheduler and the calibrated output pulse sequences. This output is easily generated for both the `QuantumCircuit` and `Schedule` with the `draw` method.

Qiskit Pulse users may create pulse programs to replace the default pulse programs of the native gate set provided by the backend and pass them as an argument to the scheduler. This gives users low level control over the gate definitions used at scheduling time. They may specify their own scheduling policies to dynamically aggregate gates and generate composite pulse sequences such as would be required to implement the compilation techniques described by Shi et. al. [25].

(a)

```
qc = QuantumCircuit(2, 2)
qc.h(1)
qc.cx(1, 0)
qc.measure([0, 1], [0, 1])
qc = transpile(qc, backend)
pulse_schedule = schedule(qc, backend)

# Plot the program representations.
qc.draw()
pulse_schedule.draw()
```

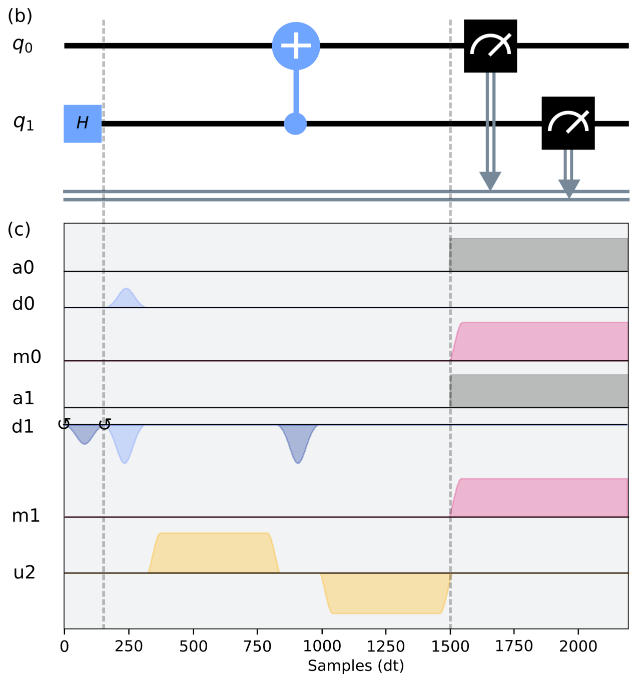

FIG. 2: (a) Qiskit code to construct a quantum circuit that prepares and measures a Bell state and then schedules the circuit to produce an equivalent pulse schedule. Here, `h` is a Hadamard gate, `cx` a CNOT gate and `backend` is a description of a quantum system received from a hardware provider. (b) and (c) Visualization of the mapping between circuit instructions (b) and the composite pulse sequences that will implement the circuit elements (c). Pulse envelopes filled with bright and dark colors respectively represent the real (in-phase) and imaginary (quadrature-phase) components of the input control waveform. The circular arrows represent a phase shift. The gray shadow on `a0` and `a1` indicates the data acquisition trigger for the ADC which is synchronized with the measurement stimulus pulse. These mappings are automatically provided by the hardware backend, but may be overridden by the user as we demonstrate in section III. The scheduler will align the gates in time according to the selected scheduling policy.

III. DEMONSTRATION OF A CROSS-RESONANCE ENTANGLING GATE

To highlight how Qiskit Pulse can enable tasks that cannot be done in the circuit model we perform standard quantum process tomography (QPT) [26] of both echoed and un-echoed cross-resonance (CR) [27] pulses for *varying amplitudes* on a cloud-based quantum computer. We use the tomography data to calculate the coefficients of the effective CR Hamiltonian as a function of the pulse amplitude, and show how to implement a high-fidelity Controlled-NOT (CNOT) gate based on the calibrated CR pulse.

A. The Cross-Resonance Interaction

The CR gate is a microwave-only two-qubit entangling gate for fixed-frequency dispersively coupled qubits [27]. It is physically realized by driving the control qubit with microwave pulses at the frequency of the target qubit to stimulate the evolution of an effective ZX interaction Hamiltonian, where Z and X are the Pauli-Z and X operators of the driven control qubit and the target qubit, respectively.

The two-transmon system driven by the CR pulse is described by a time-dependent Hamiltonian $H_{\rm CR}(t)$ which, in the absence of noise, results in the unitary evolution $U_{\rm CR}$. We can approximate the evolution as being generated by a time-independent Hamiltonian $\overline{H}_{\rm CR}$ using the perturbative method presented in Ref [28] which showed good agreement with experimental results for single-pulse and echoed CR gates [24]. This technique approximates the time-dependent control qubit drive pulse with a constant amplitude pulse and block-diagonalizes the resulting Hamiltonian to second order. Using this approach the CR evolution is approximated by $U_{\rm CR} \approx \exp\left(-it_{\rm CR}\overline{H}_{\rm CR}\right)$ where

$$\begin{aligned} \overline{H}_{\rm CR} &= \frac{Z\otimes B}{2} + \frac{I\otimes C}{2} \qquad (2)\\ B &= \omega_{ZI} I + \omega_{ZX} X + \omega_{ZY} Y + \omega_{ZZ} Z \\ C &= \omega_{IX} X + \omega_{IY} Y + \omega_{IZ} Z. \end{aligned}$$

If the ZX term could be isolated, the resulting unitary gate would be a two-qubit rotation

$$U_{\rm ZX}(\theta_{ZX}) = \exp\left(-i\theta_{ZX}\frac{ZX}{2}\right), \qquad (3)$$

where the rotation angle θ_{ZX} depends on the strength and duration of the pulse applied to the control qubit. The unitary gate $U_{\rm ZX}(\pi/2)$ is a *perfect entangler* – it can be used to generate a maximally entangled state from a separable input state and is locally equivalent to a CNOT gate [29]. Therefore, combined with arbitrary single-qubit operations, it is sufficient for universal quantum computation.

The terms in addition to $\omega_{ZX}ZX$ in $\overline{H}_{\rm CR}$ lead to coherent errors and divergences from the ideal target unitary in Eq. (3). Characterizing the strength of these terms and designing pulse sequences that suppress them is necessary to create high fidelity entangling operations. The standard techniques used to suppress these terms are multi-pulse echos and cancellation tones [24].

B. Constructing and Calibrating a Cross-Resonance Gate

The experiments presented within this section are executed on the twenty-qubit IBM Quantum system `ibmq_almaden` to take advantage of higher resolution waveforms with a cycle-time `dt` $= 0.222$ ns afforded by infrastructure under test on that system at the time of writing. We use qubit 1 and qubit 0 as the control and target qubits, respectively. The resonance frequency and anharmonicity of the control qubit are $f_1 = 4.972$ GHz and $\delta_1 = -319.7$ MHz, and $f_0 = 4.857$ GHz and $\delta_0 = -320.2$ MHz for the target qubit.

We implement both a single-pulse (CR1), and an echoed two-pulse (CR2) variant of the CR gate without a cross-talk cancellation tone on the target qubit [30]. The CR pulse envelope is a `GaussianSquare` pulse, i.e. a square pulse with Gaussian-shaped rising and falling edges. The pulse has a square amplitude A, a phase $\phi = -0.166$ rad., discussed in Appendix B, and a total duration $t_{\rm CR} = 848$ `dt` $= 184.4$ ns. The square portion of the pulse has a duration of 720 `dt` and the Gaussian rising and falling edges last 64 `dt` and have a 32 `dt` standard deviation. The pulse duration is chosen so that a $\pi/2$ rotation angle can be achieved within the weak driving regime.

The CR1 sequence is a single CR pulse on the `ControlChannel u1`, see Fig. 3(a). The CR2 sequence consists of two CR pulses with opposite phases on `u1`, and two additional single-qubit pulses on the `DriveChannel d1`, one after each CR pulse, see Fig. 3(b). This echo sequence refocuses unwanted terms in the interaction Hamiltonian [24]. The following code exemplifies how to build the CR2 schedule in Qiskit Pulse.

```
# Create a pulse schedule
sched = Schedule(name='cr2')

# Create pulse objects for the echoed CR gate
cr_p = GaussianSquare(cr_dur, amp, sigma, square_width)
cr_m = GaussianSquare(cr_dur, -amp, sigma, square_width)
x180 = Drag(pi_dur, pi_amp, pi_sigma, pi_beta)

# Assemble the schedule
sched += Play(cr_p, ControlChannel(1))
sched += Delay(t_cr, DriveChannel(1))
sched += Play(x180, DriveChannel(1))
sched += Play(cr_m, ControlChannel(1))
sched += Delay(t_cr, DriveChannel(1))
sched += Play(x180, DriveChannel(1))
```

C. Quantum Process Tomography of the CR Gate

To study the dynamics of the CR pulse, we perform standard QPT [26] of the CR1 and CR2 pulse sequences for a range of CR pulse amplitudes using the tomography module of Qiskit Ignis [31]. Given a d-dimensional noisy

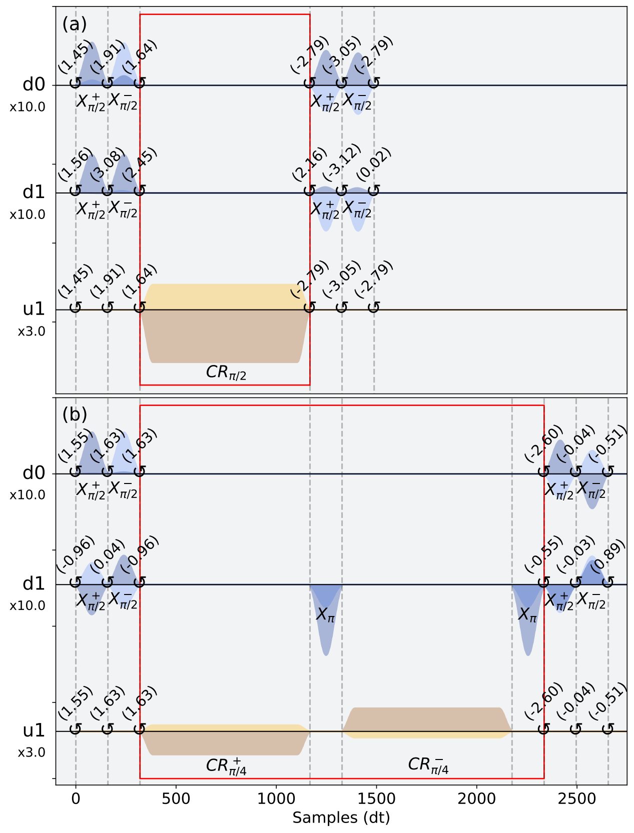

FIG. 3: CNOT pulse `Schedules` implemented by the CR gates with the local fidelity optimization. Local operations are realized by $X^{\pm}_{\pi/2}$ pulses with three virtual-Z gates before and after CR gates on the `DriveChannels` `d0` and `d1`. The CR1 and CR2 gate are surrounded by red boxes. (a) CR1-based CNOT gate composed of a single CR pulse $CR_{\pi/2}$ on the `ControlChannel` `u1`. (b) CR2-based CNOT gate composed of two CR pulses $CR^{\pm}_{\pi/4}$ on `u1` with echo pulses X^+_π applied on `d1`. Measurement and acquisition pulses are not shown. The numbers below the channel aliases show an amplitude scaling factor used for plotting. The 12 circular arrows topped by floating point numbers represent phase shifts in units of radians and each phase shift corresponds to an optimization parameter Θ_i. Note that phase shifts on `u1` reflect those in `d0` to synchronize the frame of both channels; they are automatically inserted by the pulse scheduler.

quantum channel $\mathcal{E}$, QPT reconstructs the *Choi*-matrix $\Lambda_\mathcal{E}$ which is the positive-semidefinite matrix defined by $\Lambda_\mathcal{E} \equiv \sum_{i=0}^{d-1} |i\rangle\langle i| \otimes \mathcal{E}(|i\rangle\langle i|)$ [32].

The QPT circuits, shown in Fig. 4, use the single-qubit gates $\{U_i^{\text{prep}}\}_{i=0}^3$ and $\{U_i^{\text{meas}}\}_{i=0}^2$ to prepare the required input states and measurement bases, respectively. We prepare each qubit in the states $|0\rangle, |1\rangle, \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle)$ and measure in the X, Y and Z bases. Our amplitude-dependent CR pulse is inserted into the QPT circuits in Fig. 4 as a custom gate that the Qiskit pulse scheduler maps to a pulse program, see appendix A 1. Each of the 144 two-qubit QPT pulse schedules is executed 2048 times to estimate the measurement outcome probabilities of each qubit. The details of the readout process are presented in Sec. IV. We correct for measurement errors using the readout error mitigation techniques [33] implemented in Qiskit Ignis. Readout error mitigation for two qubits requires four additional schedules which were interleaved with the QPT schedules. The reconstructed Choi-matrix $\mathcal{E}_{\text{CR}}(A)$ for the noisy gate was obtained from the convex-optimization QPT fitter in Qiskit Ignis for each value of the CR pulse amplitude A. This fitter uses maximum likelihood estimation to find the completely-positive trace-preserving process that is most likely to fit the measured data after correction for readout errors.

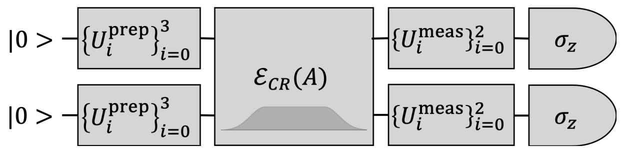
|0 >
|0 >
$\{U_i^{\text{prep}}\}_{i=0}^3$
$\{U_i^{\text{prep}}\}_{i=0}^3$
$\mathcal{E}_{CR}(A)$
$\{U_i^{\text{meas}}\}_{i=0}^2$
$\{U_i^{\text{meas}}\}_{i=0}^2$
σ_z
σ_z

FIG. 4: Process tomography circuits for the $\mathcal{E}_{\text{CR}}(A)$ pulse embedded as a user-defined custom gate.

We use the fitted Choi-matrices $\mathcal{E}_{\text{CR}}(A)$ to compute estimates of the coefficients of the effective CR Hamiltonian in Eq. (2). The method is described in Appendix C. We then fit these coefficients to a third order model to find the CR pulse amplitude that implements a $\theta_{ZX} = \pi/2$ rotation, see appendix C. The estimated amplitudes, marked by the stars in Fig. 5, were 0.229 ± 0019 and 0.098 ± 0005 for CR1 and CR2, respectivly.

D. Optimizing CNOT Fidelity with Local Operations

To estimate the highest fidelity of a maximally entangling gate that the CR gate can be transformed into, we optimize the average gate fidelity F over all single-qubit pre and post-rotation angles Θ on both the control and target qubits, see appendix D. The optimized fidelities for the measured CR1 and CR2 process maps are $F_{\max} = 0.992$ and $F_{\max} = 0.994$.

We then use the Qiskit transpiler and pulse scheduler to optimize and build a CNOT gate from the calibrated CR gate and the device-calibrated single-qubit gates [34] which implement the optimal local rotation parameters Θ from Eq. (D1). The optimized CNOT schedules are shown in Fig. 3. The average gate fidelities of the calibrated CNOT gates are measured with randomized benchmarking (RB) [35]. The details of the pulse program applying the local rotations and setting up the RB measurements are given in Appendix A 2. The RB experiments estimate an average gate fidelity of $F = 0.981$ and $F = 0.979$ for the CR1 and CR2 gates, respectively. These fidelities are comparable to the measured fidelity of $F = 0.984$ of the standard CNOT gate provided by `ibmq_almaden` which is implemented using a highly-tuned

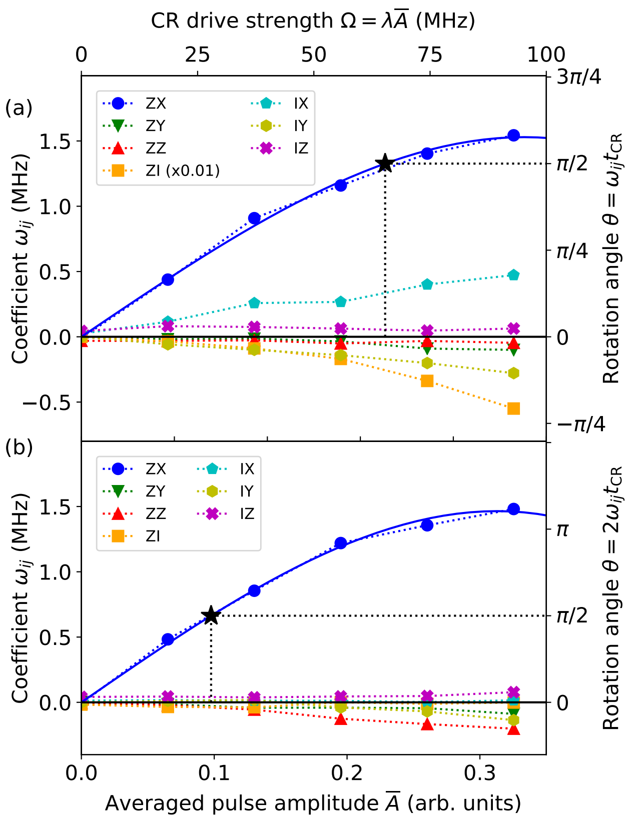

FIG. 5: CR Hamiltonian coefficients reconstructed following the procedure described in Appendix C, plotted against the time-averaged CR pulse amplitude $\overline{A}$. The blue solid lines are the fits to the third order expansion of the CR Hamiltonian in Eq. (C6). The rotation angle is estimated by $\theta = n_{CR}\omega_{ij}t_{\text{CR}}$ where n_{CR} is the number of CR pulses. Star-shaped markers indicate operating points of CNOT gates. (a) Measured coefficients of the CR1 sequence. The ZI coefficient is several orders of magnitude larger than other Hamiltonian terms and is displayed with a 1/100 scale in (a) whereas in (b) it is approximately zeroed by the echo pulse in CR2 and is therefore not scaled.

calibration process including an echo sequence, cancellation tone, and closed-loop amplitude calibration [24]. It is worth noting that the CNOT gates demonstrated in this paper have no cancellation tone and all parameters are obtained with open-loop calibration. In the same way, we can create custom basis gates which may enable hardware-efficient implementations of quantum algorigthms.

IV. READOUT AT THE PULSE LEVEL

Readout is the process through which the qubit state is projected onto $|0\rangle$ or $|1\rangle$ and a corresponding classical bit is obtained. This process is modeled by a readout chain in which the observed signal undergoes a series of successive transformations. Qiskit Pulse supports returning the output data of each measurement layer to the programmer. The lowest level accessible to the user, level-zero or *raw* data, typically corresponds to a digitized time-series signal. A kernel method applied to the signal data removes its time dependency and results in a complex value which encodes the qubit state (level-one *kerneled* data). Finally, the classified qubit state (level-two *disciminated* data), is obtained by applying a discriminator to the kerneled data. For a superconducting qubit processor, the time traces are complex vectors representing the digitized readout signals reflected or transmitted from the readout resonators [9]. The kernel method, such as the boxcar integrator used within this paper, outputs points in the IQ plane which a discriminator may use to classify the qubit's state.

Qiskit users are now able to retrieve data from different levels in the readout-chain by specifying the readout data-level, i.e., zero, one, or two. For example, users of IBM Quantum processors may request the kerneled data in the form of IQ points so that they may implement their own discriminator. To be sure, Qiskit users that do not wish to implement their own kernels and discriminators can request level-two data therefore using the built-in readout scheme. The readout methodology that we implemented reflects the typical data flow during readout in hardware and should allow users to test novel readout schemes [36] as well as accommodate different quantum computing architectures.

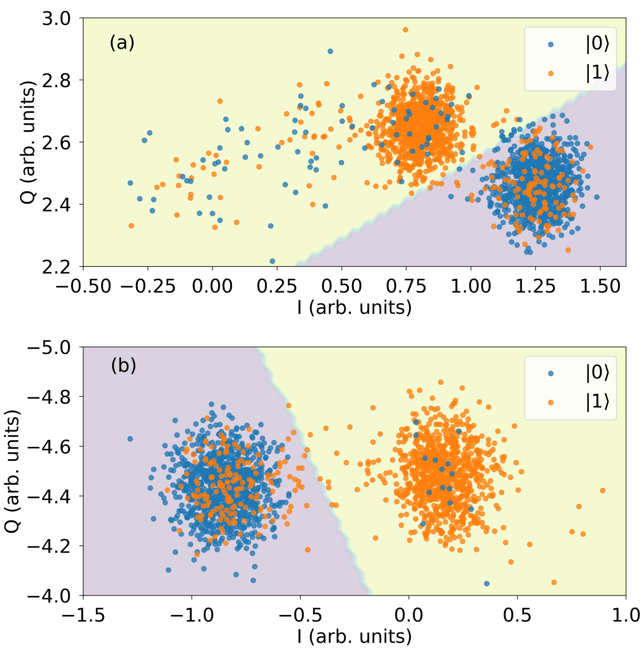

FIG. 6: IQ data and the decision boundary for the discriminators of qubits 16, shown in (a), and 17, shown in (b), that use only the IQ data of their respective qubit.

Aside from counting qubit states, discriminators may also be used to infer properties of the system and benchmark it. We illustrate this by investigating spurious correlations in the qubit readout of the IBM Quantum sys-

TABLE II: The measurement assignment fidelity $F_a = 1 - (\Pr[0|1] + \Pr[1|0])/2$ of the four discriminators [36]. For example, the Single-Q16 discriminator was fitted with the calibration schedules `cal_00` and `cal_01` while the discriminator Both-Q16 was fitted with all four calibration schedules: `cal_00`, `cal_01`, `cal_10` and `cal_11`. The confidence intervals were obtained using Jeffreys interval at a 95% confidence level.

		Qubit discriminated	
		Q16	Q17
Data used	Single	$89.21^{+1.78}_{-2.00}\%$	$91.06^{+1.42}_{-1.65}\%$
	Both	$89.48^{+1.26}_{-1.38}\%$	$90.62^{+1.05}_{-1.16}\%$

tem `ibmq_singapore` selected based on its availability at the time of the experiment. From `ibmq_singapore` we randomly selected qubits 16 and 17 to study as they are a neighboring pair of qubits on the chip. Kerneled data was measured for four calibration schedules, named `cal_ij` with $i, j \in \{0, 1\}$. Here, to prepare the state $|ij\rangle$ a π-pulse is applied to qubit i if $i = 1$ and simultaneously to qubit j if $j = 1$. The single-qubit pulses are followed by the measurement stimulus pulses and acquisition instructions for all qubits. For each qubit we fit two discriminators based on linear discriminant analysis [36]. For qubit i, one discriminator is fitted to a subset of the calibration data in which the other qubit is always in state $|0\rangle$, shown in Fig. 6, while the other discriminator is fitted using all four calibration schedules. We expect that the discriminator that uses all the calibration data will perform best if there is measurement cross-talk. The fidelities of the fitted discriminators, shown in Tab. II, suggest that there is no significant cross-talk between the qubits that we measured. This is verified by t-tests on sixteen Pearsons' correlation coefficients $r_j(ES_{i,X}, GS_{i,Y})$ between $ES_{i,X}$ and $GS_{i,Y}$ which correspond to the $X, Y \in \{\mathrm{I}, \mathrm{Q}\}$ data of qubit i in state $j \in \{0, 1\}$ when the other qubit is in the excited state (ES, i.e., $|1\rangle$) and ground state (GS, i.e., $|0\rangle$), respectively. These correlation coefficients are sensitive to cross-talk between the two qubits. With 1024 degrees of freedom, i.e., measurement shots, we do not observe any statistically significant correlation at the 95% confidence level. This implies that it is sufficient to fit discriminators using only a ground and excited reference schedule for each qubit and consequently only $2n$ calibration schedules are required for n qubits when there is no cross-talk rather than the 2^n calibration schedules that would be required with all-to-all measurement crosstalk.

V. CONCLUSIONS AND FUTURE WORK

Rapid development in quantum computing has led to publicly available quantum computers with an increasing number of qubits, improved connectivity, and greater control. Prior to this work, publicly available quantum programming frameworks for cloud-based quantum computers have been at the relatively high-level of the circuit model, or implementation-specific, thus limiting their application. In the near-term, pulse-level control is desired to extract as much quantum volume as possible from the hardware by experimenting with novel control and characterization schemes [37–39].

We have introduced Qiskit Pulse, an implementation of the virtual pulse-level programming model, OpenPulse [8]. We have demonstrated that the Qiskit circuit scheduler can target pulse instructions and that physical superconducting qubit hardware can interpret these instructions to execute useful programs. By embedding our pulse programming instruction set in Qiskit we have integrated gate-level quantum programs and classical pulse stimulus, exposing a new level of hardware control to Qiskit users. The benefit that pulse control provides quantum programmers was demonstrated by calibrating a cross-resonance pulse on a cloud-based quantum computer and embedding it as a gate within the standard circuit programming model and characterizing this user-defined gate using quantum process tomography.

Giving users pulse-level access to current-day quantum computers will allow them to explore techniques such as error mitigation and dynamical decoupling schemes that cannot be investigated at the circuit level. In the future we will explore embedding the pulse programming model as a coprocessor within a classical virtual instruction set architecture that supports classical arithmetic and control-flow [40]. We will also investigate extensions to the pulse programming model such as defining special purpose registers to track phase across multiple channels which would reduce the number of required `PulseChannels` and enable simpler tracking of shared phase for composite gates. We would then use these capabilities to explore the implementation of active error-correcting codes, and promising variational quantum-classical algorithms such as the variational quantum eigensolver [41–43].

ACKNOWLEDGEMENTS

We thank Blake Johnson for insightful discussions. We thank Andrew Wack and John Lemek for providing system access and deployments.

[1] Héctor Abraham and et al. Qiskit: An open-source framework for quantum computing, 2019.

[2] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[3] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open quantum assembly language, 2017.

[4] Isaac Chuang. Qasm2circ. https://www.media.mit.edu/quanta/qasm2circ/, Mar 2005.

[5] N. Khammassi, G. G. Guerreschi, I. Ashraf, J. W. Hogaboam, C. G. Almudever, and K. Bertels. cQASM v1.0: Towards a common quantum assembly language, 2018.

[6] Steffen J. Glaser, Ugo Boscain, Tommaso Calarco, Christiane P. Koch, Walter Köckenberger, Ronnie Kosloff, Ilya Kuprov, Burkhard Luy, Sophie Schirmer, Thomas Schulte-Herbrüggen, Dominique Sugny, and Frank K. Wilhelm. Training schrödinger's cat: quantum optimal control. *The European Physical Journal D*, 69(12):279, Dec 2015.

[7] Jens Koch, Terri M. Yu, Jay Gambetta, A. A. Houck, D. I. Schuster, J. Majer, Alexandre Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf. Charge-insensitive qubit design derived from the cooper pair box. *Phys. Rev. A*, 76:042319, Oct 2007.

[8] David C. McKay, Thomas Alexander, Luciano Bello, Michael J. Biercuk, Lev Bishop, Jiayin Chen, Jerry M. Chow, Antonio D. Corcoles, Daniel J. Egger, Stefan Filipp, Juan Gomez, Michael Hush, Ali Javadi-Abhari, Diego Moreda, Paul Nation, Brent Paulovicks, Erick Winston, Christopher J. Wood, James Wootton, and Jay M. Gambetta. Qiskit backend specifications for OpenQASM and OpenPulse experiments, 2018.

[9] Philip Krantz, Morten Kjaergaard, Fei Yan, Terry P. Orlando, Simon Gustavsson, and William D. Oliver. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6, 2019.

[10] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 813–825, New York, NY, USA, 2017. ACM.

[11] Anastasiia Butko, George Michelogiannakis, Samuel Williams, Costin Iancu, David Donofrio, John Shalf, Jonathan Carter, and Irfan Siddiqi. Understanding quantum control processor capabilities and limitations through circuit characterization, 2019.

[12] Colm A. Ryan, Blake R. Johnson, Diego Rist, Brian Donovan, and Thomas A. Ohki. Hardware for dynamic quantum computing. *Review of Scientific Instruments*, 88(10):104703, 2017.

[13] Thomas Alexander, Naoki Kanazawa, Daniel J. Egger, Lauren Capelluto, Christopher J. Wood, Ali Javadi-Abhari, and David McKay. Notebooks and Data for "Qiskit-Pulse: Programming Quantum Computers Through the Cloud with Pulses". February 2020. doi: 10.5281/zenodo.3751565.

[14] Tzvetan S. Metodi, Darshan D. Thaker, Andrew W. Cross, Frederic T. Chong, and Isaac L. Chuang. Scheduling physical operations in a quantum information processor. In *Quantum Information and Computation IV*, volume 6244, page 62440T. International Society for Optics and Photonics, May 2006.

[15] Navin Khaneja, Timo Reiss, Cindie Kehlet, Thomas Schulte-Herbrggen, and Steffen J. Glaser. Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms. *Journal of Magnetic Resonance*, 172(2):296 – 305, 2005.

[16] Kristan Temme, Sergey Bravyi, and Jay M. Gambetta. Error mitigation for short-depth quantum circuits. *Phys. Rev. Lett.*, 119:180509, Nov 2017.

[17] Lewis Fry Richardson and J. Arthur Gaunt. VIII. The deferred approach to the limit. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 226(636-646):299–361, January 1927.

[18] Abhinav Kandala, Kristan Temme, Antonio D. Córcoles, Antonio Mezzacapo, Jerry M. Chow, and Jay M. Gambetta. Error mitigation extends the computational reach of a noisy quantum processor. *Nature*, 567:491–495, 2019.

[19] David J. Lurie. Numerical design of composite radiofrequency pulses. *Journal of Magnetic Resonance (1969)*, 70(1):11–20, October 1986.

[20] David C. McKay, Christopher J. Wood, Sarah Sheldon, Jerry M. Chow, and Jay M. Gambetta. Efficient Z gates for quantum computing. *Phys. Rev. A*, 96:022330, Aug 2017.

[21] 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. 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. *Phys. Rev. Lett.*, 112:190504, May 2014.

[22] Johannes Heinsoo, Christian Kraglund Andersen, Ants Remm, Sebastian Krinner, Theodore Walter, Yves Salathé, Simone Gasparinetti, Jean-Claude Besse, Anton Potočnik, Andreas Wallraff, and Christopher Eichler. Rapid high-fidelity multiplexed readout of superconducting qubits. *Phys. Rev. Applied*, 10:034040, Sep 2018.

[23] Keith Cooper and Linda Torczon. *Engineering a Compiler*. Morgan Kaufmann, San Francisco, 1 edition edition, November 2003.

[24] Sarah Sheldon, Easwar Magesan, Jerry M. Chow, and Jay M. Gambetta. Procedure for systematically tuning up cross-talk in the cross-resonance gate. *Phys. Rev. A*, 93:060302, Jun 2016.

[25] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I. Schuster, Henry Hoffmann, and Frederic T. Chong. Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 1031–1044, Providence, RI, USA, April 2019. Association for Computing Machinery.

[26] M. Mohseni, A. T. Rezakhani, and D. A. Lidar. Quantum-process tomography: Resource analysis of dif-

ferent strategies. *Phys. Rev. A*, 77:032322, Mar 2008.

[27] Jerry M. Chow, A. D. Córcoles, Jay M. Gambetta, Chad Rigetti, B. R. Johnson, John A. Smolin, J. R. Rozen, George A. Keefe, Mary B. Rothwell, Mark B. Ketchen, and M. Steffen. Simple all-microwave entangling gate for fixed-frequency superconducting qubits. *Phys. Rev. Lett.*, 107:080502, Aug 2011.

[28] Easwar Magesan and Jay M. Gambetta. Effective hamiltonian models of the cross-resonance gate, 2018.

[29] Jun Zhang, Jiri Vala, Shankar Sastry, and K. Birgitta Whaley. Geometric theory of nonlocal two-qubit operations. *Phys. Rev. A*, 67:042313, Apr 2003.

[30] D. Willsch, M. Nocon, F. Jin, H. De Raedt, and K. Michielsen. Gate-error analysis in simulations of quantum computers with transmon qubits. *Phys. Rev. A*, 96(6):062302, 2017.

[31] IBM. Qiskit Ignis. https://github.com/Qiskit/qiskit-ignis, 2019. Accessed: 2019-12-31.

[32] Christopher J. Wood, Jacob D. Biamonte, and David G. Cory. Tensor networks and graphical calculus for open quantum systems. *Quant. Inf. Comp.*, 15:0579–0811, 2015.

[33] A Dewes, FR Ong, V Schmitt, R Lauro, N Boulant, P Bertet, D Vion, and D Esteve. Characterization of a two-transmon processor with individual single-shot qubit readout. *Phys. Rev. Lett.*, 108(5):057002, 2012.

[34] The general single-qubit gate is built from a pair of $\pi/2$-pulses with three virtual Z-gates with rotation angles Θ_i, Θ_j, and Θ_k.

[35] Easwar Magesan, J. M. Gambetta, and Joseph Emerson. Scalable and Robust Randomized Benchmarking of Quantum Processes. *Phys. Rev. Lett.*, 106(18):180504, May 2011.

[36] Easwar Magesan, Jay M. Gambetta, A. D. Córcoles, and Jerry M. Chow. Machine learning for discriminating quantum measurement trajectories and improving readout. *Phys. Rev. Lett.*, 114:200501, May 2015.

[37] Lev Bishop, Sergey Bravyi, Andrew W. Cross, Jay M. Gambetta, and John A. Smolin. Quantum Volume. 2017.

[38] Gregory A. L. White, Charles D. Hill, and Lloyd C. L. Hollenberg. Performance optimisation for drift-robust fidelity improvement of two-qubit gates, 2019.

[39] Max Werninghaus, Daniel J. Egger, Federico Roy, Shai Machnes, Frank K. Wilhelm, and Stefan Filipp. Leakage reduction in fast superconducting qubit gates via optimal control, 2020.

[40] Vikram Adve, Chris Lattner, Michael Brukman, Anand Shukla, and Brian Gaeke. LLVA: A low-level virtual instruction set architecture. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 205–, Washington, DC, USA, 2003. IEEE Computer Society.

[41] Kyungjoo Noh and Christopher Chamberland. Fault-tolerant bosonic quantum error correction with the surface-GKP code, 2019.

[42] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242246, Sep 2017.

[43] M. Ganzhorn, D.J. Egger, P. Barkoutsos, P. Ollitrault, G. Salis, N. Moll, M. Roth, A. Fuhrer, P. Mueller, S. Woerner, I. Tavernelli, and S. Filipp. Gate-efficient simulation of molecular eigenstates on a quantum computer. *Phys. Rev. Applied*, 11:044092, Apr 2019.

[44] Sebastian Krinner, Simon Storz, Philipp Kurpiers, Paul Magnard, Johannes Heinsoo, Raphael Keller, Janis Luetolf, Christopher Eichler, and Andreas Wallraff. Engineering cryogenic setups for 100-qubit scale superconducting circuit systems. *EPJ Quantum Technology*, 6(1):2, 2019.

Appendix A: Programming with Qiskit Pulse

1. Performing Quantum Process Tomography

The code example below demonstrates how the quantum process tomography (QPT) schedules for the CR1 used in Sec. III are created with Qiskit Pulse.

```python
from qiskit import transpile, schedule
from qiskit.circuit import QuantumRegister, QuantumCircuit, Gate
from qiskit.pulse import Schedule, Play, ControlChannel
from qiskit.pulse.pulse_lib import GaussianSquare
from qiskit.ignis.verification.tomography import process_tomography_circuits
from qiskit.test.mock import FakeAlmaden

# Unique name for the instruction.
gate_name = 'cr1'

# Pulse parameters determined during calibration.
duration = 848
sigma = 32
square_width = 64
amp = 0.2

```

```
# Call the backend and get basis_gates and inst_map.
backend = FakeAlmaden()
config = backend.configuration()
defaults = backend.defaults()
basis_gates = config.basis_gates
inst_map = defaults.instruction_schedule_map

# Create the CR1 schedule.
cr1_pulse = GaussianSquare(duration, amp, sigma, square_width)
sched = Schedule()
sched += Play(cr1_pulse, ControlChannel(0))

# Add the CR1 instruction to basis_gates and inst_map.
basis_gates += [gate_name]
inst_map.add(gate_name, [1, 0], sched)

# Create a quantum gate to reference the CR1 pulse schedule.
cr1_gate = Gate(gate_name, 2, [])

# Create the QPT circuits.
qregs = QuantumRegister(config.n_qubits)
circuit = QuantumCircuit(qregs)
circuit.append(cr1_gate, qargs=[qregs[1], qregs[0]])
qpt_circuits = process_tomography_circuits(circuit, [qregs[0], qregs[1]])

# Create the QPT pulse schedules.
qpt_circuits = transpile(qpt_circuits, backend, basis_gates)
qpt_schedules = schedule(qpt_circuits, backend, inst_map)
```

The above example uses the mock backend `FakeAlmaden` for IBM Quantum system `ibmq_almaden` which can be substituted for the real backend.

The pulse envelope of the CR1 `cr1_pulse` is created with a flat-topped Gaussian pulse. The pulse schedule `sched` of CR1 is then added to the `basis_gates` and the circuit instruction to pulse schedule mapping (`inst_map`) for qubits one and zero with the name `cr1`. The `basis_gates` defines a list of primitive circuit instructions available in the system and the `inst_map` defines a lookup table of calibrated pulse schedules for each basis gate on each qubit. The `circuit` object of the CR1 sequence is created with a custom gate `cr1_gate`. The QPT circuits are then assembled by calling the `process_tomography_circuits` function in Qiskit-Ignis. This appends state preparation circuits and measurement circuits before and after the `circuit`. The returned `qpt_circuits` is a list of quantum circuits containing each possible combination of input states and measurement bases. The `qpt_circuits` are then mapped to the backend in question by Qiskit's transpiler, taking into account the extended set of `basis_gates`. Finally, we call the pulse scheduler with the custom `inst_map` containing CR1 to create the QPT pulse schedules. The QPT program for the CR2 sequence is created with the same procedure.

2. Performing Randomized Benchmarking

The code example below demonstrates how the standard randomized benchmarking (RB) schedules with the CR1-CNOT used in Sec. III D are created in the qiskit pulse module.

```
from qiskit import transpile, schedule
from qiskit.circuit import QuantumCircuit, Gate
from qiskit.pulse import Schedule, Play, ControlChannel
from qiskit.pulse.pulse_lib import GaussianSquare
from qiskit.ignis.verification.randomized_benchmarking import randomized_benchmarking_seq
from qiskit.test.mock import FakeAlmaden

# Unique name for instruction.
```

```python
gate_name = 'cr1'

# Randomized benchmark setup.
n_seed = 5
n_clifford = [1, 21, 51, 101, 151]

# Pulse parameters determined during calibration.
duration = 848
sigma = 32
square_width = 64
amp = 0.2

# Local rotation angles for CNOT(1,0) determined during optimization.
local_rotations10 = [[1.45, 1.91, 1.64],
                     [1.56, 3.08, 2.45],
                     [-2.79, -3.05, -2.79],
                     [2.16, -3.12, 0.02]]

# Local rotation angles for CNOT(0,1) determined during optimization.
local_rotations01 = [[-1.68, 3.04, 1.66],
                     [1.57, 2.28, -0.06],
                     [1.48, -0.46, 3.14],
                     [1.60, -3.14, 0.98]]

# Call the backend and get the basis_gates and inst_map.
backend = FakeAlmaden()
config = backend.configuration()
defaults = backend.defaults()
basis_gates = config.basis_gates
inst_map = defaults.instruction_schedule_map

# Create the CR1 schedule.
cr1_pulse = gaussian_square(duration, amp, sigma, square_width)
sched = Schedule()
sched += Play(cr1_pulse, ControlChannel(0))

# Add the CR1 instruction to basis_gates and inst_map.
basis_gates += [gate_name]
inst_map.add(gate_name, [1, 0], sched)

# Create a quantum gate to reference the CR1 pulse schedule.
cr1_gate = Gate(gate_name, 2, [])

# Build a CNOT(1,0) schedule based on CR1 schedule.
qregs = QuantumRegister(config.n_qubits)
cnot10 = QuantumCircuit(qregs)
cnot10.u3(*local_rotations10[0], qregs[0])
cnot10.u3(*local_rotations10[1], qregs[1])
cnot10.append(cr1_gate, qargs=[qregs[1], qregs[0]])
cnot10.u3(*local_rotations10[2], qregs[0])
cnot10.u3(*local_rotations10[3], qregs[1])

cnot10 = transpile(cnot10, backend, basis_gates)
cnot_sched10 = schedule(cnot10, backend, inst_map)

# Build a CNOT(0,1) schedule based on CR1 schedule.
qregs = QuantumRegister(config.n_qubits)
cnot01 = QuantumCircuit(qregs)
```

```python
cnot01.u3(*local_rotations01[0], qregs[0])
cnot01.u3(*local_rotations01[1], qregs[1])
cnot01.append(cr1_gate, qargs=[qregs[1], qregs[0]])
cnot01.u3(*local_rotations01[2], qregs[0])
cnot01.u3(*local_rotations01[3], qregs[1])

cnot01 = transpile(cnot01, backend, basis_gates)
cnot_sched01 = schedule(cnot01, backend, inst_map)

# Overwrite the default CNOT schedule in the inst_map.
inst_map.add('cx', [1, 0], cnot_sched10)
inst_map.add('cx', [0, 1], cnot_sched01)

# Create randomized benchmarking circuits with 5 seeds.
rb_circuits_seeds, _ = randomized_benchmarking_seq(n_seed, n_clifford, [[0, 1]])

# Schedule the randomized benchmarking experiment into pulse schedules.
rb_schedules_seeds = []
for rb_circuits_seed in rb_circuits_seeds:
    rb_circuits_seed = transpile(rb_circuits_seed, backend, basis_gates)
    rb_schedules_seed = schedule(rb_circuits_seed, backend, inst_map)
    rb_schedules_seeds.append(rb_schedules_seed)
```

As shown in Sec. A 1, the pulse schedules are programmed with the aid of the `QuantumCircuit` class to apply device calibrated single-qubit gates around the CR1 pulse sequence abstracted by `cr1_gate`. The local rotation parameters can be obtained by the optimization routine shown in Sec. D. It should be noted that in a two qubit standard RB sequence the CNOT gate can assign both qubit 0 and 1 as a control qubit. Beacuse the CNOT gate is not identical under the exchange of the control and the target qubits , we need to prepare pulse schedules for both qubit arrangements. Then, the default CNOT instruction in the `inst_map` is overwitten by the pulse schedules based on the calibrated CR1 sequence. Finally, the RB circuits are generated by a call to the `randomized_benchmarking_seq` function in Qiskit-Ignis. The returned `rb_circuits_seeds` is a list of RB circuits for each random seed. These RB circuits are then independently transpiled and scheduled to create RB pulse programs. The RB programs for the CR2 sequence are created with the same procedure.

Appendix B: Cross Resonance Phase Calibration

In the twenty-qubit IBM Quantum system `ibmq_almaden`, microwave pulses programmed with Qiskit Pulse are generated by waveform generators at room temperature and travel through coaxial cables to the qubits [44]. The transfer function between the room temperature electronics and the qubits can cause a phase offset ϕ_0 in Eq. (1) resulting in an error in the rotation axis of the target qubit. The Hamiltonian may therefore have an unwanted ZY interaction term which we eliminate by adjusting the phase of CR pulse ϕ. We perform this calibration with the CR2 schedule since its time-independent Hamiltonian, which we approximate by

$$\overline{H}_{\mathrm{CR}} \simeq \Omega(A,\phi)\left(\cos\phi_0 ZX + \sin\phi_0 ZY\right) + \varepsilon, \tag{B1}$$

has less terms than the CR1 Hamiltonian due to the echo. Here, Ω is the strength of the CR drive as a function of its amplitude A and its phase ϕ while ε represents the small interaction terms which are not fully refocused by the echo sequence.

First, we initialize the qubit in the $|00\rangle$ state. We sweep the amplitude A and measure the target qubit in the Pauli-Z basis to find the pulse amplitude $A_{\mathrm{opt}} = 0.108$ which creates an equal superposition of $|0\rangle$ and $|1\rangle$. If the offset ϕ_0 is zero this transformation is a $\pi/2$-rotation around the X-axis so that the target qubit, measured in the Y-basis, yields $\mathrm{Tr}(\hat{\sigma_{\mathrm{y}}}\,\rho) = \pm 1$ with the sign depending on the state of the control qubit. We thus measure the readout signal at A_{opt} in the Y-basis for both initial states of the control qubit $|10\rangle$ and $|00\rangle$, while sweeping the phase ϕ. The calibrated phase that maximizes $|\,\mathrm{Tr}(\hat{\sigma_{\mathrm{y}}}\,\rho)|$ is $\phi_{\mathrm{opt}} = -0.166$ rad.

Appendix C: Effective Hamiltonian Estimation and Amplitude Calibration

We use the fitted Choi-matrices $\mathcal{E}_{\rm CR}(A)$ to compute estimates of the coefficients of the effective CR Hamiltonian in Eq. (2). Since a real CR pulse will have noise the resulting process is not unitary. Noisy quantum evolution for a time-independent Hamiltonian in the presence of Markovian noise may be described by the *Lindblad equation* $\frac{d}{dt}\rho(t) = \mathcal{G}(\rho)$ with the Lindblad generator

$$\mathcal{G}(\rho) = \mathcal{L}_H(\rho) + \mathcal{D}(\rho) \tag{C1}$$

$$\mathcal{L}_H(\rho) = -i[H,\rho] \tag{C2}$$

$$\mathcal{D}(\rho) = \sum_j \gamma_j \left(A_j \rho A_j^\dagger - \frac{1}{2}\{A_j^\dagger A_j, \rho\}\right), \tag{C3}$$

where the operator $\mathcal{L}$ is the generator of the unitary evolution, and $\mathcal{D}$ is the generator of the non-unitary dissipative evolution. As with unitary evolution, the Lindblad equation can be solved as a matrix differential equation obtaining $|\rho(t)\rangle\rangle = S_{\mathcal{E}}|\rho(0)\rangle\rangle$, where $|A\rangle\rangle$ denotes a *column-vectorized* matrix A, and $S_{\mathcal{E}} = \exp(tS_{\mathcal{G}})$ is the *superoperator* representation of quantum process $\mathcal{E}$ [32].

For a Hamiltonian H we note that the operators $B_{ij} = \frac{1}{2}P_i \otimes P_j$, with P_i are single-qubit Pauli operators, define an orthonormal basis for two-qubit operators — i.e. $\mathrm{Tr}[B_{ij}B_{kl}^\dagger] = \delta_{ik}\delta_{jl}$. Hence for a Hamiltonian given by $H = \sum_{ij}\omega_{ij}B_{ij}$, we can extract the coefficients via $\omega_{ij} = Tr[B_{ij}^\dagger H]$. The superoperator for the Hamiltonian component of $\mathcal{G}$ is given by

$$S_{\mathcal{L}_H} = -i(\mathbb{I}\otimes H - H^T \otimes \mathbb{I}). \tag{C4}$$

We can use the fact that the superoperators of the Hamiltonian basis term $S_{\mathcal{L}_{B_{ij}}}$ are also an orthogonal (but not-normalized) basis for $S_{\mathcal{L}_H}$, and importantly, are orthogonal to the dissipative part of the generator ($\mathrm{Tr}[S_{\mathcal{L}_{B_{ij}}}^\dagger S_{\mathcal{D}}] = 0$) when the dissipator only involves Pauli and T_1 and T_2 relaxation terms. This allows us to extract the coefficients from the Lindblad superoperator generator as

$$\omega_{ij} = \frac{\mathrm{Tr}\left[S_{\mathcal{L}_{B_{ij}}}^\dagger S_{\mathcal{G}}\right]}{\|S_{\mathcal{L}_{B_{ij}}}\|}. \tag{C5}$$

To compute the superoperator generator $S_{\mathcal{G}}$, we first obtain the Choi-matrix estimate for a channel $\mathcal{E}$ from quantum process tomography and then convert it to the superoperator representation $S_{\mathcal{E}}$. For additional details on the superoperators and converting between superoperators and the Choi-matrix representation obtained from tomography see [32]. Next, we take the matrix logarithm to obtain the generator $S_{\mathcal{G}} = t^{-1}\log(S_{\mathcal{E}})$ from which we estimate ω_{ij} for our two-qubit system using Eq. (C5). The process fidelities of the estimated CR Hamiltonian using this technique and the experimentally obtained Choi-matrix are 99.4 % and 98.6 % on average for CR1 and CR2 experiments, respectively.

We find that as predicted only the terms ZX, ZY, ZZ, ZI, IX, IY, and IZ, shown in Fig. 5, are significant for CR1 and CR2 while all other remaining Pauli terms are negligible. In both CR sequences the ZY term is suppressed by the calibrated CR phase $\phi_{\rm opt}$ and a monotonic increase of the desired ZX term is observed as the pulse amplitude A increases. The CR1 pulse without echoing has large contributions from the IX, IY and ZI terms, see Fig. 5(a). Such unwanted interactions, except for ZZ, are removed by the echo sequence in CR2, compare Fig. 5(a) and (b). The effect of these unwanted interactions can be reduced by applying single-qubit gates before and after the CR pulse to correct for local coherent errors as discussed in Sec. III D.

We now find the CR pulse amplitude that creates a maximum entangling gate, i.e. $\theta_{ZX} = \pi/2$ in Eq. (3). Due to the Gaussian edges of our CR pulses we relate the drive strength Ω to the time-averaged pulse amplitude $\overline{A}$ through a linear response $\Omega = \lambda\overline{A}$. The measured ZX interaction strengths are fit by the third order expansion of the CR Hamiltonian [28]

$$\begin{aligned}\frac{\omega_{ZX}(\overline{A})}{2} =& -\frac{J\lambda\overline{A}}{\Delta}\left(\frac{\delta_1}{\delta_1+\Delta}\right)\\ &+\frac{J(\lambda\overline{A})^3\delta_1^2(3\delta_1^3+11\delta_1^2\Delta+15\delta_1\Delta^2+9\Delta^3)}{4\Delta^3(\delta_1+\Delta)^3(\delta_1+2\Delta)(3\delta_1+2\Delta)}\end{aligned} \tag{C6}$$

where J is the coupling strength, δ_1 is the anharmonicity of the control qubit, and Δ is the frequency difference between the control and target qubits. In this model, we have a pair of fit parameters J and λ. The coupling strength obtained from the fit was $J = 1.87 \pm 0.046$ MHz and 1.79 ± 0.033 MHz for the CR1 and CR2 data, respectively. The λ coefficient was -271.2 ± 12.2 MHz and -288.9 ± 10.2 MHz for the CR1 and CR2 data, respectively. These fit parameters are independent of the pulse sequences and both results almost agree within the error range. The small mismatch between the fit values may be caused by imperfections in the Hamiltonian reconstructed from the tomography data. These fit curves yield the controlled rotation angle as a function of the CR pulse amplitude $\theta_{ZX}(\overline{A}) = n_{CR}\omega_{ZX}(\overline{A})t_{\mathrm{CR}}$ where $n_{CR} = 1$ for CR1 and $n_{CR} = 2$ for CR2. Finally, we can find the pulse amplitudes $\overline{A}$ for $\theta_{ZX} = \pi/2$. The estimated amplitudes were respectivly 0.229 ± 0019 and 0.098 ± 0005 for CR1 and CR2. These are marked by the stars in Fig. 5. Due to the nonlinearity between the ZX term and the average pulse amplitude $\overline{A}$, see Eq. (C6), the estimated drive amplitude of CR1 is slightly larger than double the drive amplitude of CR2.

Appendix D: Optimizing CNOT Fidelity with Local Operations

To estimate the highest fidelity of a maximally entangling gate that the CR gate can be transformed into, we optimize the average gate fidelity F over all single-qubit pre and post-rotations on both the control and target qubits:

$$F_{\mathrm{max}} = \max_{\Theta} F[\mathcal{E}_{CR}(\overline{A}_{\pi/2}), U_{\mathrm{ent}}(\Theta)] \tag{D1}$$

where the optimization is over 12 real parameters for the four parameterized $U_3(\Theta_i, \Theta_j, \Theta_k)$ rotations in $SU(2)$:

$$\begin{aligned} U_{\mathrm{ent}}(\Theta) &= U^{\dagger}_{\mathrm{pre}}(\Theta) U_{\mathrm{ent}} U^{\dagger}_{\mathrm{post}}(\Theta), \\ U_{\mathrm{pre}}(\Theta) &= U_3(\Theta_0, \Theta_1, \Theta_2) \otimes U_3(\Theta_3, \Theta_4, \Theta_5), \\ U_{\mathrm{post}}(\Theta) &= U_3(\Theta_6, \Theta_7, \Theta_8) \otimes U_3(\Theta_9, \Theta_{10}, \Theta_{11}). \end{aligned}$$

The ideal unitary matrix for the cross-resonance perfect-entangler is $U_{\mathrm{ent}} = ZX(\pi/2)$ and for the CNOT gate is $U_{\mathrm{ent}} = CX$. This optimization aims to remove the effect of locally correctable coherent errors. It will, however, underestimate the error of the transformed CR gate as it neglects errors in the single-qubit gates. The optimized fidelities for the measured CR1 and CR2 process maps are $F_{\mathrm{max}} = 0.994$ and $F_{\mathrm{max}} = 0.998$.