CERN-TH-2024-126

An open-source framework for quantum hardware control

Edoardo Pedicillo[1,2], Alessandro Candido[3], Stavros Efthymiou[1], Hayk Sargsyan[1], Yuanzheng Paul Tan[6], Juan Cereijo[1], Jun Yong Khoo[4], Andrea Pasquale[1,2], Matteo Robbiati[2,5] and Stefano Carrazza[1,2,3]

[1]Quantum Research Center, Technology Innovation Institute, Abu Dhabi, UAE
[2]TIF Lab, Dipartimento di Fisica, Università degli Studi di Milano
[3]Theoretical Physics Department, CERN, 1211 Geneva 23, Switzerland
[4]Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), 1 Fusionopolis Way, #16-16 Connexis, Singapore 138632, Singapore
[5]European Organization for Nuclear Research (CERN), Geneva 1211, Switzerland
[6]Division of Physics and Applied Physics, School of Physical and Mathematical Sciences, Nanyang Technological University, 21 Nanyang Link, Singapore 637371, Singapore

Abstract.

The development of quantum computers needs reliable quantum hardware and tailored software for controlling electronics specific to various quantum platforms. Middleware is a type of computer software program that aims to provide standardized software tools across the entire pipeline, from high-level execution of quantum computing algorithms to low-level driver instructions tailored to specific experimental setups, including instruments. This paper presents updates to `Qibolab`, a software library that leverages `Qibo`'s capabilities to execute quantum algorithms on self-hosted quantum hardware platforms. `Qibolab` offers an application programming interface (API) for instrument control through arbitrary pulses and driver operations including sweepers. This paper offers an overview of the new features implemented in `Qibolab` since Ref. [1], including the redefined boundaries between platform and channel classes, the integration of an emulator for simulating quantum hardware behaviour, and it shows updated execution times benchmarks for superconducting single qubit calibration routines.

1 Introduction

Nowadays, quantum computing research and development needs both reliable quantum hardware and classical hardware. The latter is what is usually called the control electronics and takes care of synthesizing the simultaneous synchronized control waveforms, specific to each different quantum platform technology, required to operate quantum hardware. The objective of middleware is to provide standardized software tools for the whole pipeline, e.g., from the high-level execution of quantum computing algorithms based on the quantum circuit paradigm, to operate the low-level control instrument instructions tailored to a specific experimental setup.

The commitment to build a software with a platform-agnostic interface helps the transition from theory to experiments by reducing the effort and expertise required to operate a quantum platform and develop novel quantum algorithms. Additionally, it ensures reusability and this allows the creation of a community of quantum laboratories that can share experiments and data significantly reducing the burden coming from the different setups.

Our commitment to open-source and community-driven software started with `Qibo` [2], a framework for gate-based and adiabatic quantum computing with hardware acceleration.

In this paper, we present updates to `Qibolab` [1], a software library that leverages `Qibo` 's potential to execute quantum algorithms on self-hosted quantum hardware platforms.

`Qibolab` takes care of all the necessary operations to prepare the execution of quantum circuits on a fully characterized device.

`Qibolab` is running on a host computer, which communicates, typically via a network protocol, with the control electronics used for generation of synchronized signal sequences. These electronics are connected to the quantum processing unit (QPU) via different channels.

In the case of operating a superconducting quantum device: the readout and feedback channels in a closed loop for measuring the qubit, the drive channels for applying gates and, for flux-tunable qubits, the flux channels for tuning their frequency. Moreover, some architectures there may have coupler qubits between some computational qubits that need flux pulses to control computational qubit interactions.

The library offers a dedicated application programming interface (API) not only for quantum circuit design, but also qubit calibration, instrument control through arbitrary pulses, driver operations as sweepers.

Thanks to `Qibolab`, we can operate the control electronics required to fully calibrate a quantum device performing specific experiments. In order to make this process as smooth as possible, we have also developed on top of `Qibolab` the `Qibocal` [3] library collecting some calibration protocols for superconducting qubits with a user-friendly interface.

Successful implementation of the whole `Qibo` ecosystem will provide the research community with a prototype of an extensible, quantum hardware-agnostic, open-source hybrid quantum operating system, fully tested and benchmarked on superconducting platforms.

In the following sections, we first give an overview of the project in Sec. 2, highlighting the differences with the old software version described in Ref. [1], then we provide the updated time benchmarks in Sec. 3, at the end we provide some details about the emulator in Sec. 4.

2 Project Overview

`Qibolab` provides four main interface objects: the `Pulse` object, which is used to define arbitrary pulses played on the quantum device, a series of pulses can be collected in `PulseSequence` and executed on a specific QPU through the `Channels`.

Pulses constitute the basic building blocks of programs executed on quantum hardware. They represent a physical pulse, i.e., they are used to read the state of a qubit, drive it to change its state, or flux bias a qubit to change its resonant frequency to probe two-qubit interactions.

`Qibolab` provides pulse objects for each one of these operations as its `Pulse` object holds the required information about amplitude, duration and phase of the pulse for the generation of physical pulses. Differently from the previous version, the `Pulse` object now further facilitates the integration with different control electronics.

Abstract pulse sequences defined using the `Pulse` API can be deployed on hardware using a `Platform`. `Platform` is the `Qibolab` core object and it orchestrates the different instruments for qubit control. Each `Platform` instance corresponds to a specific quantum device controlled by a specific set of instruments. It allows users to execute a single sequence, a batch of sequences, or perform a sweep, in which one or more pulse parameters are updated in real-time, within the control instrument without external communication. This new `Qibolab` version allows the `Platform` to execute all of them within a single instrument connection step, increasing the uptime of the QPU.

The `Platform` object also contains single qubit, two qubit interactions and coupler qubits information and eventually any other quantum components like parametric amplifiers that may be present in the experimental setup. Since, computational qubits and coupling qubits are similar objects from the instrument point of view, as elements you send pulses to, so in the software level they can be defined by the same class we called `QuantumElement`.

To define two qubit interactions, `QubitPair` objects contain information about coupled qubits pairs in a given device for a given configuration and their corresponding two-qubit native gates.

Finally, `Channel` represents the connection between the quantum device inputs and outputs to the proper instrument port.This connection is essential for the instrument our interface to target the desired `QuantumElement`. It also provides a `QuantumElement`-centric interface instrument parameter setting, which is useful in calibration routines.

`Channels` are also responsible of signal generation. Differently from the previous `Qibolab` version, now the readout is decoupled into probe and acquisition channels, thus giving more freedom to the end user to decide which part of the readout pulse to acquire or extend the acquisition beyond the time interval defined by the readout pulse.

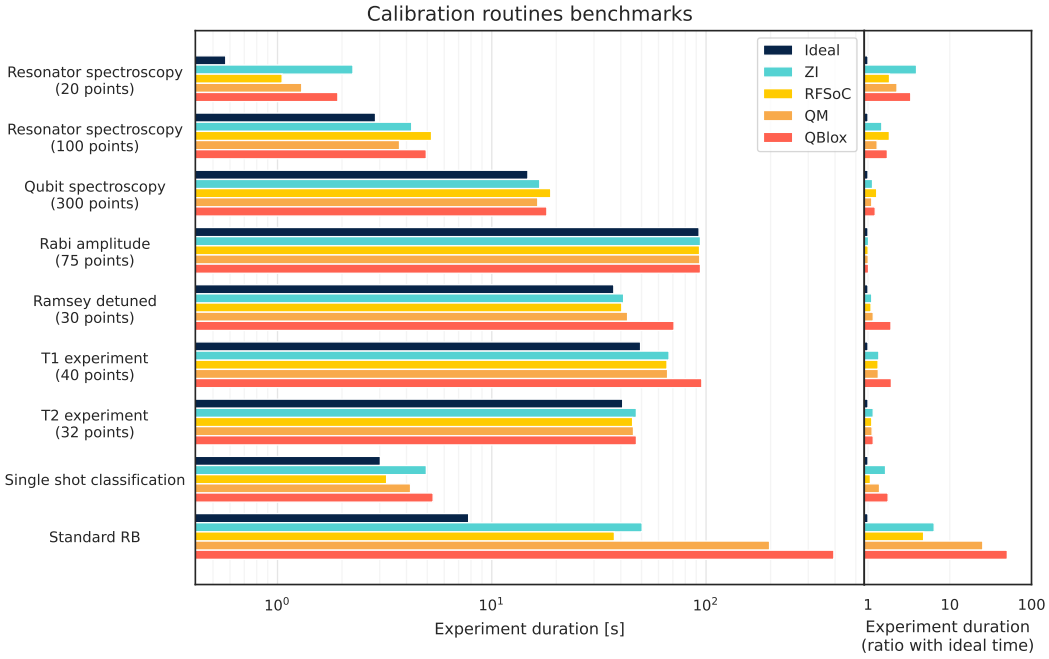

Figure 1: Execution time of different qubit calibration routines on various electronics as in Ref. [1]. On the left side, we show the absolute times in seconds for each experiment. The ideal time (black bar) shows the minimum time the qubit needs to be affected in each experiment. On the right side, we calculate the ratio between the actual execution time and the ideal time. Real-time sweepers are used, if supported by the control device, in all cases except the *Ramsey detuned* and *Standard RB* experiments.

With the new refactoring, the boundaries between the `Platform` and `Channel` classes are redefined: there is no more distinction between readout and drive pulses. Since, they are just defined by the physical channel where the pulses are sent, the frequency is now a `Channel` property, so this class is closer to a logical definition of channel.

Another significant change is `Transpiler` which got moved to `Qibo`. This choice is driven by a rethinking of the respective roles of the `Qibo` ecosystem libraries and transparency. The transpilation procedure consists of processing the sequence of gates required by an algorithm and integrate SWAP gates, if necessary, so that the connectivity of the chip is respected. In practice, given the topology of the device, this is a task completely solvable within `Qibo` itself. Once a connectivity-compatible sequence of gates is defined, `Qibolab` provides a compiler, that is, the translation of abstract gates into native gates, and thus into pulses.

3 Cross-platform benchmark

In this section, we show the updated benchmarks already presented in the `Qibolab` paper. They are related to the execution times of a set of calibration experiments performed on a single qubit by different control electronics (see Tab.1) currently supported by `Qibolab` out of the box.

In contrast to the previous benchmark version, `Qibolab` side, we have implemented the possibility to unroll a list of sequences to a single pulse sequence that contains multiple measurements. This approach achieves much faster execution compared to executing the sequences one by one in a software loop. On the side of the control electronics, we have updated the Zurich Instrument software to `LabOneQ 2.16.0` [4].

The experiments chosen for this benchmark represent the minimal set of routines required for superconducting single qubit calibration.

They also offer a view of the different execution modes supported by `Qibolab`: in particular the *Single shot classification* experiment executes fixed pulses sequences, while the *Spectroscopies* perform different sweeps over pulse parameters.

All the results are summarized in Fig. 1, for each calibration protocol we provide also the theoretical execution time T_{ideal}, defined as the total duration of the pulse sequences executed during the acquisition,

Device	Firmware	Software
Qblox	0.4.0	qblox-instruments 0.9.0 [5]
QM	QOP213	qm-qua 1.1.1 [6]
Zurich	Latest (November 2023)	LabOneQ 2.16.0 [4]
RFSoCs	Qick 0.2.135 [7]	Qibosoq 0.0.3 [8]
Erasynth++	-	-
R&S SGS100A	-	QCoDeS 0.37.0 [9]

Table 1: Outline of the devices and firmware/software version supported during the benchmark.

whose formula [1] is

$$\mathrm{T}_{\mathrm{ideal}} = n_{\mathrm{shots}} \sum_i (T_{\mathrm{sequence},i} + T_{\mathrm{relaxation}}). \tag{1}$$

where $T_{\mathrm{sequence},i}$ is the duration of the whole pulse sequence in the i-th point of the sweep, $T_{\mathrm{relaxation}}$ the time we wait for the qubit to relax to its ground state between experiments, n_{shots} the number of shots in each experiment and the sum runs over all points in the sweep. The ideal time denotes how long the qubit is used during an experiment and provides the baseline for our benchmark.

As already pointed out in Ref. [1], the comparison between the ideal and real execution times show that the first one is always less than the second one, because of different overheads, indeed we can express the real execution time $\mathrm{T}_{\mathrm{real}}$ as

$$\mathrm{T}_{\mathrm{real}} = T_{\mathrm{qibo}} + T_{\mathrm{inst}} + \mathrm{T}_{\mathrm{ideal}}, \tag{2}$$

where T_{qibo} is the overhead coming from `Qibolab` backend and T_{inst} the instruments one. We found that the overhead coming from the `Qibolab`, T_{qibo}, is negligible compared to that of the control instruments, i.e.,

$$\mathrm{T}_{\mathrm{real}} \simeq T_{\mathrm{inst}} + \mathrm{T}_{\mathrm{ideal}}. \tag{3}$$

4 Emulator

The new `Qibolab` version supports emulators to simulate quantum hardware. It is a crucial tool especially for `Qibocal`, as it could be used both for testing the software itself (especially when access to the real device is limited or unavailable) and serve as a digital twin to a specific quantum hardware to assist with a range of functionalities including predicting output of calibration experiments, coarse-grain calibrations, pulse-shaping and test-bedding, and more. For the latter, the emulator requires the device parameters that characterize the quantum hardware of interest as inputs to build an accurate model of the hardware.

To integrate the emulator into the `Qibolab` ecosystem, we deployed an ad-hoc controller class called `PulseSimulator`. The `PulseSimulator` is exclusively called by the emulator and it serves primarily as a middleman to translate and communicate objects between `Qibolab` and the selected quantum dynamics simulation library (hereafter referred to as the simulation engine) used to numerically solve the underlying quantum dynamics of the device model in the presence of time-dependent control pulse sequences. Specifically, it initializes the simulation engine with a device model specified by the device parameters and simulation settings in the runcard, extracts modulated signal waveforms from `Qibolab` pulse sequences and sends them to the simulation engine which in turn performs the dynamics simulation, and at the end of the simulation, translates the results generated from the simulation engine back to `Qibolab` (or `Qibo`) result objects.

From the user's point of view therefore, the emulator platform behaves like a quantum hardware platform and both are equivalent in terms of attributes and functions. Consequently, it requires a platform folder and it is initialized as any other device platform. In addition, the emulator returns simulated results – a time series of statevectors (or density matrices when including dissipation) of the underlying quantum system that was generated sequentially by the simulation engine as it solves the dynamics described by the Lindblad master equation,

$$\dot{\rho}(t) = -i[H(t), \rho(t)] + \sum_k \frac{\gamma_k}{2} \left[2A_k \rho(t) A_k^\dagger - \rho(t) A_k^\dagger A_k - A_k^\dagger A_k \rho(t) \right]. \tag{4}$$

In the above, $\rho(t)$ is the density matrix of the system's quantum state at time t, $H(t)$ is its time-dependent Hamiltonian, and A_k are the operators through which the environment couples to the system with rate γ_k. As `Qibolab` currently only supports superconducting-qubits-based hardware, the device model used by the emulator is based on N capacitively coupled transmon qubits modelled as Duffing oscillators [10, 11] with number of energy levels predefined by the user in the `PulseSimulator` settings:

$$H(t) = H_{\text{sys}} + H_{\text{drive}}(t), \tag{5}$$

$$H_{\text{sys}} = \sum_{i=1}^{N} \left(\omega_i b_i^\dagger b_i + \frac{\alpha_i}{2} b_i^\dagger b_i (b_i^\dagger b_i - 1) \right) + \sum_{i \neq j} g_{ij} (b_i^\dagger b_j + b_i^\dagger b_j), \tag{6}$$

$$H_{\text{drive}}(t) = \sum_{i=1}^{N} \left[\Omega_{X,i}(t) \cos(\omega_{\text{drive},i} t) + \Omega_{Y,i}(t) \sin(\omega_{\text{drive},i} t) \right] (b_i^\dagger + b_i). \tag{7}$$

In the above, $\omega_i, \omega_{\text{drive},i}$ and α_i denote the resonant frequency, drive frequency and anharmonicity respectively for transmon i, $b_i(b_i^\dagger)$ its annihilation(creation) operator, and $\Omega_{X,i}(t), \Omega_{Y,i}(t)$ the drive amplitudes on its quadratures, while g_{ij} denotes the coupling strength between transmon i and j. Decoherence is incorporated using the Bloch-Redfield model [10], whereby each transmon is characterized by its longitudinal and transverse relaxation times $T_{i,1}$ and $T_{i,2}$ respectively,

$$A_{i,1} = \frac{1}{2}(\sigma_{i,X} + i\sigma_{i,Y}), \quad A_{i,2} = \sigma_{i,Z}, \quad \gamma_{i,1(2)} = \frac{2\pi}{T_{i,1(2)}}, \tag{8}$$

with $\sigma_{i,\mu=X,Y,Z}$ the Pauli matrices corresponding to transmon i. As an example, we show in Fig. 2 the state overlap between the transmon qubit, modelled as a three-level system, with each of its energy modes as it interacts with the pulse for the X gate followed by the Hadamard gate. This can be easily extended to include other quantum computing technologies when they are available on `Qibolab`. The first supported simulation engine is based on `QuTiP` [12], with plans to incorporate `JAX` [13] support for GPU acceleration, as well as tensor-network based quantum dynamics simulation libraries to speed up the simulation of larger system sizes with a lower memory footprint but with a small accuracy cost.

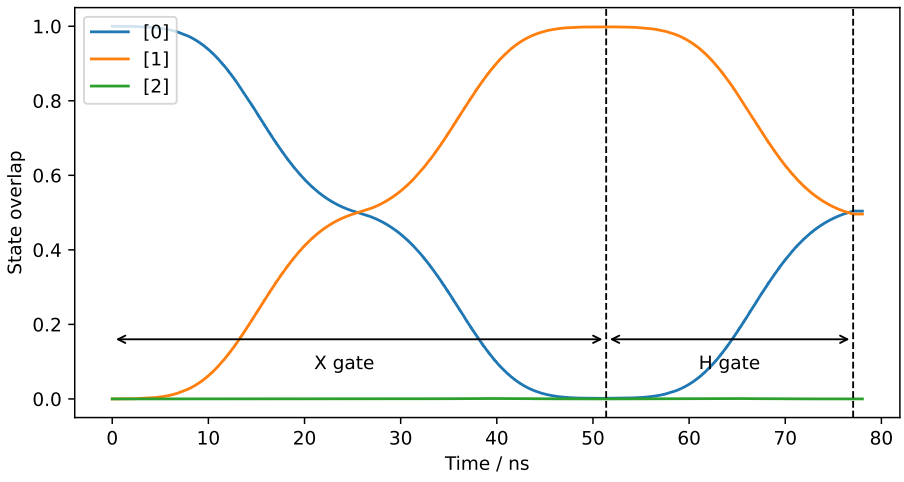

Figure 2: State overlap between the simulated qubit modelled as a three-level system with each of its energy modes as it evolves under a control pulse sequence for an X gate followed by a Hadamard (H) gate.

For the initial release, resonators are not included in the simulation model and therefore the only acquisition modes supported are the discrimination and integration, where the latter is currently implemented as a projection onto the in-phase pulse component. Despite this limitation, the available emulator can already execute most `Qibocal` protocols to simulate calibration and device characterization. This

part of the library is still a work in progress and we plan to expand it with new features. Increasing the complexity of the simulated quantum system with the inclusion of flux tunable qubits, resonators and couplers is part of our roadmap.

5 Conclusions

In this proceedings, we have presented significant updates to `Qibolab`, a software library designed to harness the full potential of `Qibo` for executing quantum algorithms on self-hosted quantum hardware platforms. The enhancements introduced in `Qibolab` include a more platform-agnostic approach to pulse definitions, streamlined orchestration of quantum hardware control through the `Platform` object, and the integration of an emulator for simulating quantum hardware behaviour.

Our work goal is a platform-agnostic interface to facilitate the transition from theoretical quantum computing models to practical experiments. This approach not only reduces the expertise required to operate diverse quantum platforms but it also ensures tools reusability as well as experiment and data sharing across a community of quantum laboratories. This collaborative environment significantly lower the barriers associated with different experimental setups.

Benchmark results demonstrate that `Qibolab` improvements lead to more efficient execution times for qubit calibration routines, highlighting the library's capability to optimize the performance of various quantum control electronics. The refined boundaries between the `Platform` and `Channel` classes and the relocation of the `Transpiler` to `Qibo` further streamline the software architecture, ensuring a clearer separation of roles and greater transparency.

The addition of the emulator component represents a crucial tool for both software testing and predictive analysis of calibration experiments. While currently, it is still a work in progress, the emulator sets the stage for future expansions, including more complex quantum systems and advanced acquisition modes.

In conclusion, the advancements in `Qibolab` not only enhance its functionality and efficiency but also contribute to the broader goal of creating an extensible, quantum hardware-agnostic, open-source hybrid quantum operating system. This system, fully tested and benchmarked on superconducting platforms, offers a robust foundation for ongoing research and development in the field of quantum computing. By fostering a collaborative community and providing powerful, flexible tools, `Qibolab` and `Qibo` collectively advance the frontier of quantum technology, supporting both current and future innovations in quantum research and applications.

6 Acknowledgments

This project is supported by `TII`'s Quantum Research Center. This work was further supported by the National Research Foundation Singapore, under its Quantum Engineering Programme 2.0 (National Quantum Computing Hub, NRF2021-QEP2-02-P01). The authors thank all `Qibo` contributors for helpful discussion and Dr. Ye Jun. M.R. is supported by `CERN`'s Quantum Technology Initiative (`QTI`) through the Doctoral Student Program. J.Y.K acknowledges funding support from A*STAR C230917003.

References

[1] Stavros Efthymiou, Alvaro Orgaz-Fuertes, Rodolfo Carobene, Juan Cereijo, Andrea Pasquale, Sergi Ramos-Calderer, Simone Bordoni, David Fuentes-Ruiz, Alessandro Candido, Edoardo Pedicillo, Matteo Robbiati, Yuanzheng Paul Tan, Jadwiga Wilkens, Ingo Roth, José Ignacio Latorre, and Stefano Carrazza. Qibolab: an open-source hybrid quantum operating system. *Quantum*, 8:1247, February 2024. ISSN 2521-327X. doi: 10.22331/q-2024-02-12-1247. URL `http://dx.doi.org/10.22331/q-2024-02-12-1247`.

[2] Stavros Efthymiou, Sergi Ramos-Calderer, Carlos Bravo-Prieto, Adriá n Pérez-Salinas, Diego García-Martín, Artur Garcia-Saez, José Ignacio Latorre, and Stefano Carrazza. Qibo: a framework for quantum simulation with hardware acceleration. *Quantum Science and Technology*, 7(1):015018, dec 2021. doi: 10.1088/2058-9565/ac39f5.

[3] Andrea Pasquale, Stavros Efthymiou, Sergi Ramos-Calderer, Jadwiga Wilkens, Ingo Roth, and Stefano Carrazza. Towards an open-source framework to perform quantum calibration and characterization, 2024. URL `https://arxiv.org/abs/2303.10397`.

[4] ZurichInstruments. `https://www.zhinst.com/others/en/quantum-computing-systems/labone-q` , 2023.

[5] Qblox. `https://www.qblox.com`.

[6] Lior Ella, Lorenzo Leandro, Oded Wertheim, Yoav Romach, Ramon Szmuk, Yoel Knol, Nissim Ofek, Itamar Sivan, and Yonatan Cohen. Quantum-classical processing and benchmarking at the pulse-level, 2023. URL https://doi.org/10.48550/arXiv.2303.03816.

[7] Leandro Stefanazzi, Kenneth Treptow, Neal Wilcer, Chris Stoughton, Collin Bradford, Sho Uemura, Silvia Zorzetti, Salvatore Montella, Gustavo Cancelo, Sara Sussman, Andrew Houck, Shefali Saxena, Horacio Arnaldi, Ankur Agrawal, Helin Zhang, Chunyang Ding, and David I. Schuster. The QICK (quantum instrumentation control kit): Readout and control for qubits and detectors. *Review of Scientific Instruments*, 93(4), April 2022. doi: 10.1063/5.0076249. URL https://doi.org/10.1063/5.0076249.

[8] Rodolfo Carobene, Alessandro Candido, Javier Serrano, Alvaro Orgaz-Fuertes, Andrea Giachero, and Stefano Carrazza. Qibosoq: an open-source framework for quantum circuit rfsoc programming, 2023. URL https://arxiv.org/abs/2310.05851.

[9] Qcodes. https://qcodes.github.io/Qcodes/, 2023.

[10] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6(2):021318, 06 2019. ISSN 1931-9401. doi: 10.1063/1.5089550. URL https://doi.org/10.1063/1.5089550.

[11] Easwar Magesan and Jay M. Gambetta. Effective hamiltonian models of the cross-resonance gate. *Phys. Rev. A*, 101:052308, May 2020. doi: 10.1103/PhysRevA.101.052308. URL https://link.aps.org/doi/10.1103/PhysRevA.101.052308.

[12] J.R. Johansson, P.D. Nation, and Franco Nori. Qutip 2: A python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 184(4): 1234–1240, 2013. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2012.11.019. URL https://www.sciencedirect.com/science/article/pii/S0010465512003955.

[13] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.