2024 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD) | 979-8-3315-1635-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/SISPAD62626.2024.10733335

Rapid Simulation Framework for Superconducting Qubit Readout System Inverse Design and Optimization

Albert Lu
Electrical Engineering Department
San Jose State University
San Jose, CA, USA
albert.lu@sjsu.edu

Hiu Yung Wong*
Electrical Engineering Department
San Jose State University
San Jose, CA, USA
hiuyung.wong@sjsu.edu

***Abstract*—Qubit readout is one of the most important operations in quantum computers. In superconducting quantum computers, the success of readout depends on many parameters and is difficult to optimize due to the high dimensionality of the problem. In this work, a rapid simulation framework that comprises an analytical model, a neural network (NN), and optimizers using the NN as a surrogate model is proposed. The analytical model is calibrated to the experimental result and allows rapid simulations to generate enough data to train NNs. Single and multi-objective optimizations are performed. It is shown that a better solution can be found using the optimizer than human optimization. Moreover, the framework can find designs with out-of-the-training-range parameters.**

Keywords—Quantum Computing, Superconducting Qubit, Readout, Measurement, Optimizer, Machine Learning

I. Introduction

Qubit readout is one of the most important operations in quantum computers (among qubit initialization, qubit state manipulation (i.e. quantum gates), and others) [1]. The success of most quantum algorithms depends on the readout accuracy (fidelity). In superconducting-based quantum computers, which are one of the most promising quantum computing architectures [2], qubit readout is realized by coupling a qubit to a resonator for dispersive readout [3]. The resonant frequency shifts based on the state of the qubit ($|0\rangle$ and $|1\rangle$). The shift is called the Cross-Kerr [4]. In this paper, we denote the *total shift* (difference between the $|0\rangle$ and $|1\rangle$ induced shifts) as χ (instead of 2χ) Usually, the readout pulse is applied at a frequency, F, to distinguish the state of the qubit by observing the Re/Im parts of the transmitted pulse through the resonator. The distinguishability of the $|0\rangle$ and $|1\rangle$ states depends on χ, F, the readout pulse power (P), duration (t_p), the resonator scattering matrix, and the noise from the circuits. It is important to co-optimize χ , F, P, t_p, and the resonator design to achieve the highest speed, least disturbance to neighboring qubits, and highest accuracy. However, this is a very difficult high-dimensional optimization problem. In this work, based on the framework in [3], a rapid simulation framework that comprises an analytical model of the resonator, a neural network (NN), and optimizers using the NN as a surrogate model is proposed. The analytical model is calibrated to the experiment result in [3]. This allows rapid simulations to generate enough data to train an NN to perform inverse design using an optimizer.

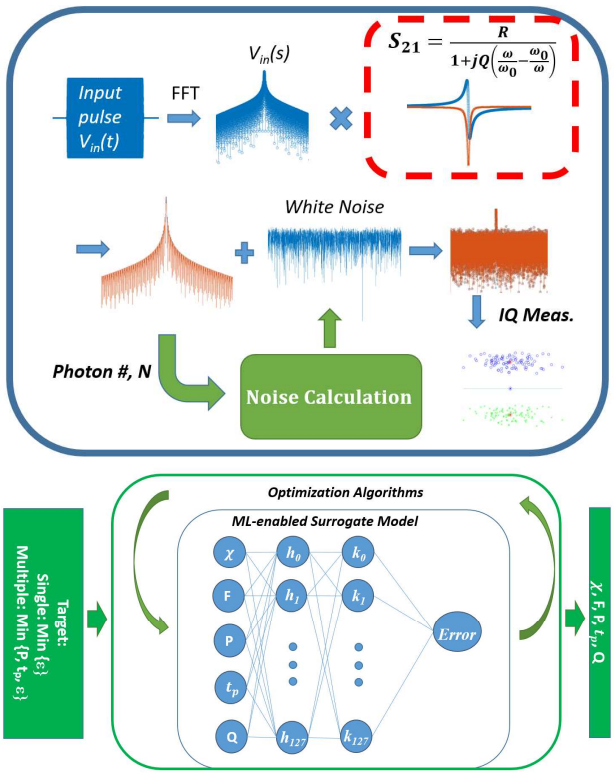

Figure 1: The optimization framework. Top: the simulation framework in [3] with S_{21} replaced by analytical calculation (red box) to rapidly generate training data. Bottom: The trained NN is used as a surrogate model for the optimization algorithm.

II. Overview and Calibration of the Framework

The quantum computer readout system being modeled is controlled by Quantum Machine OPX with a nominal readout pulse with P = -47 dBm and t_p = 3.5µs. The nominal readout

*Corresponding author: hiuyung.wong@sjsu.edu

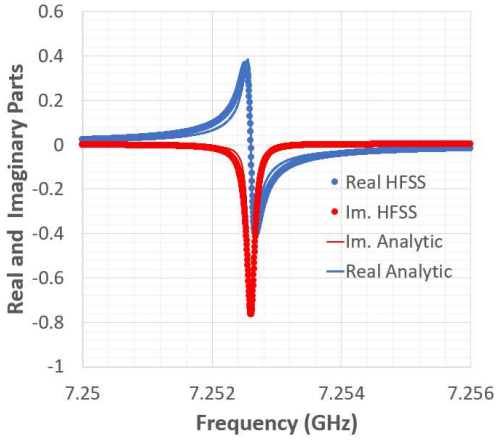

Figure 2: Comparison between the real and imaginary parts of the resonator S_{21} using HFSS and the proposed analytical model.

frequency is taken at F = 7.2525341GHz which is the mid-point of the total dispersive shift in the model in [3]. After the attenuators and due to the attenuations in the cables, the power is measured to be -123 dBm when it reaches the resonator. In the following, P is expressed relative to -123dBm. Noises from the amplifiers and quantum noises due to photon fluctuation and the quantum-limited amplifier are modeled as white noise with the corresponding noise temperatures. Particularly, the quantum noise temperature is found to be 0.5K. The details of the quantum computer and noise modeling can be found in [3].

The framework is shown in Fig. 1. Unlike [3], HFSS simulation of the resonator S_{21} is replaced by an analytical model, $S_{21} = \frac{R}{1+jQ\left(\frac{\omega}{\omega_0}-\frac{\omega_0}{\omega}\right)}$, where ω_0 is the resonant frequency. This saves hours, if not days, of simulation and design time for each resonator. To verify that this approach is valid, Fig. 2 shows that the real and imaginary parts of the 3D resonator S_{21} in [3] can be matched well with $Q = 70{,}000$ (nominal value) and $R = 0.77$. It is then used in the simulator to predict the readout error as a function of readout pulse energy (relative to -123dBm) for

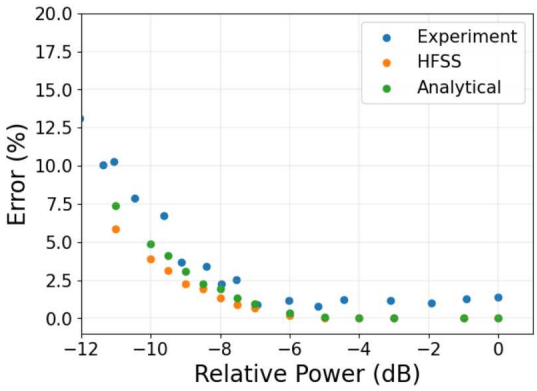

Figure 3: Comparison between experimental readout error in [3] and simulation readout errors using HFSS S_{21} and analytical S_{21} as a function of readout pulse power relative to -123dBm. Note that the non-zero error at high power (e.g. -2dB) in experiment is due to other sources such as state preparation errors which are not captured in this framework. 2000 simulations are performed for each data point.

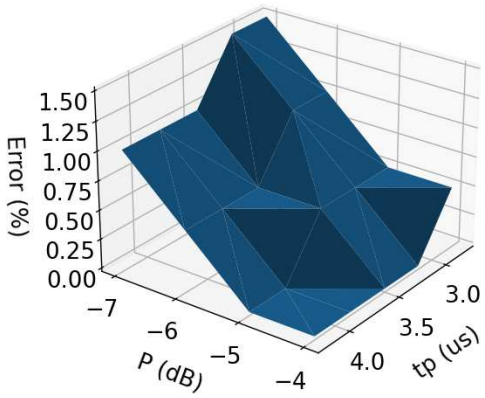

Figure 4: Readout error as a function of pulse power (P) and pulse width (t_p) when nominal Q, χ, and F are used.

χ=150,000Hz (nominal value). Fig. 3 shows that it matches the experiment and the simulator with HFSS well.

III. Data Generation

Using the new simulator with the analytical S_{21} model, 4266 simulations with different sets of parameters (χ, F, P, t_p, Q) are completed within 3 days on 100 cores. χ (±20%), $F(\pm 72kHz$ which is $\pm 40\%$ of the maximum χ), P (-4dB to -8dB), t_p ($\pm 40\%$), and Q (±20%) are varied in the simulations about the aforementioned nominal values. In each simulation, 200 runs with random noise are performed to get the readout error for the corresponding set of parameters. Plotting the errors against different variables helps enhance the understanding of the trade-offs. For example, Fig. 4 shows the error as a function of P and t_p when the nominal values of other variables are used. It shows that P = -5dB and t_p= 3.85μs is the most optimal point that can achieve a 0% error. As another example, Fig. 5 shows that the error is not monotonically dependent on the readout frequency. There is a peak around which the error is maximum for a constant power (≤ -5dB). Therefore, to explore non-trivial trends and perform optimization, NN and optimization are required.

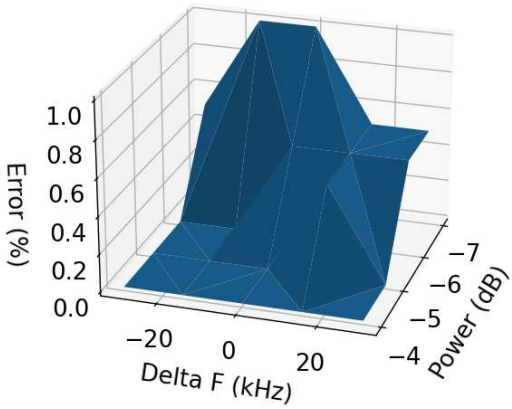

Figure 5: Readout error as a function of pulse power (P) and readout frequency (offset to the nominal value) when nominal Q and χ are used with t_p = 3.85μs.

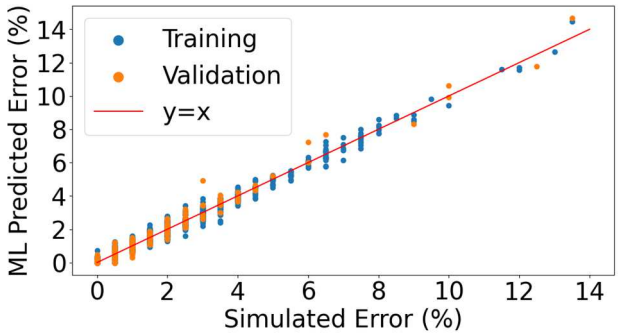

Figure 6: Comparison of the error predicted by the NN and the simulated error for both the training and validation sets.

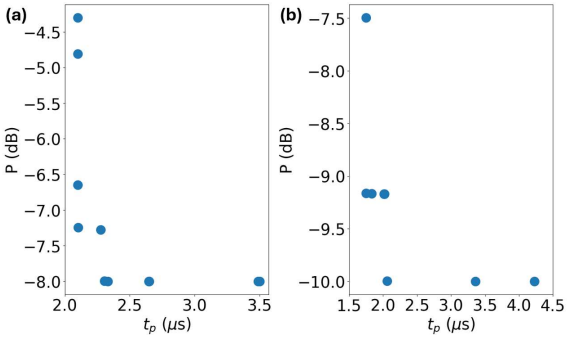

Figure 8: Multi-objective optimization solutions for bounds (a) "Within-Range" and (b) "Out-of-Range". All points represent a predicted readout error of 0.

IV. Optimization And Inverse Design

An NN is created using PyTorch [5] to correlate the 5 parameters to the error using the 4266 data points. It contains 1 input layer, 2 hidden layers (both have 128 hidden nodes with ReLU activation), and 1 output layer (ReLU activation) (Fig. 1). ReLU is used as the output activation function, instead of linear activation, to enforce the constraint of positive readout error. The Adam optimizer is used with an initial learning rate of 10^{-2} and mean squared error is used as the loss function. ReduceLROnPlateau is also used to train the model faster by decreasing the learning rate when it plateaus. The minimum learning rate is set to 10^{-5}. A batch size of 128 is used and the training is set to run for 150 epochs. To train the model and perform hyperparameter tuning, the dataset of 4266 data points is split into a train (80%), validation (10%), and test set (10%). These datasets are all normalized before input into the NN. The NN achieves an R^2 of 0.97 on the validation set and an R^2 of 0.96 on the test set (Fig. 6).

The NN is then used as a surrogate model for single-objective and multi-objective optimizations. For single-objective optimization, the sole objective is to find the parameters that will minimize the readout error. Single-objective optimization is performed using the differential evolution algorithm from SciPy [6] to perform inverse design. The differential evolution algorithm does not use gradients to find the minimum but uses a population-based search algorithm. One may limit the bounds/ranges of the parameters the optimizer can search for. As a baseline, the bounds are first set to be the same as the bounds used to generate the data (dubbed "Within-Range" which means within the parameter range of the training data). The optimizer is then run 10 times with a different seed each time to obtain a variety of possible solutions. These proposed solutions are then run in the simulation framework for verification.

An additional study is further conducted which allows the optimizer to search outside the parameter range of the training data (dubbed "Out-of-Range"). The bounds used in the "Out-of-Range" study are set as χ (±30%), F(±108kHz), P (-10dB to -2dB), t_p (±50%), and Q (±30%). In the study, each of the five parameters is allowed to have an "Out-of-Range" search while the other four are limited to the "Within-Range" search. Then each optimizer run consists of running the differential evolution algorithm 10 times with a different seed each time.

The results of "Within-Range" and "Out-of-Range" are shown in Fig. 7. It plots the difference between ML/optimizer error prediction and simulation error. It can be seen that when the optimizer is allowed to search outside of the training data

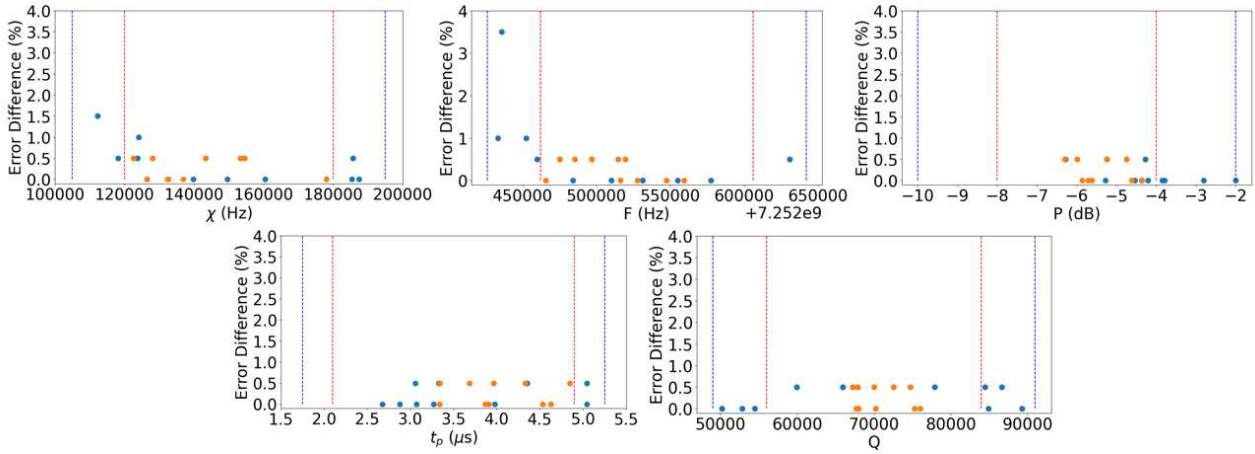

Figure 7: Single-objective optimization results of the error difference between ML and simulation against a desired parameter. Orange: Within-Range prediction. Blue: Out-of-Range prediction. Vertical dashed lines show the parameter ranges (Red: Within-Range; Blue: Out-of-Range).

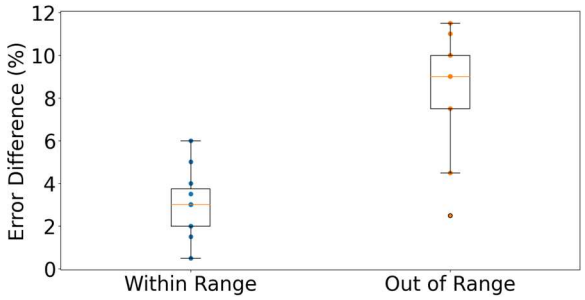

Figure 9: Multi-objective optimization results for the readout error difference between simulation and ML/optimizer when the optimizer bounds are set either within the training range or out of the training range.

parameter ranges, it can still find good designs (0% error) with "Out-of-Range" parameters.

Multi-objective optimization (minimizing P, t_p, and readout error concurrently) is then performed to determine if other conditions could be met that would lead to a more performant design. The multi-objective optimization is performed using NSGA-II from the Pymoo library [7]. NSGA-II (Non-dominated Sorting Genetic Algorithm II) is a type of multi-objective evolutionary algorithm [8]. The population size is set initially to 1000 and the algorithm is run for 200 generations. Both "Within-Range" and "Out-of-Range" optimizations are conducted as the single-objective optimization. However, in the "Out-of-Range" case, all parameters are allowed to be searched out of the training range simultaneously. The optimization algorithm is then run, and many solutions are obtained that either prioritize minimizing power, width, readout error, or all of them. Fig. 8 shows the predicted solutions obtained from optimization and how certain points are more optimized to certain criteria than others. Note that these points all represent a predicted readout error of 0, but the optimization is able to also predict points with higher readout error if desired.

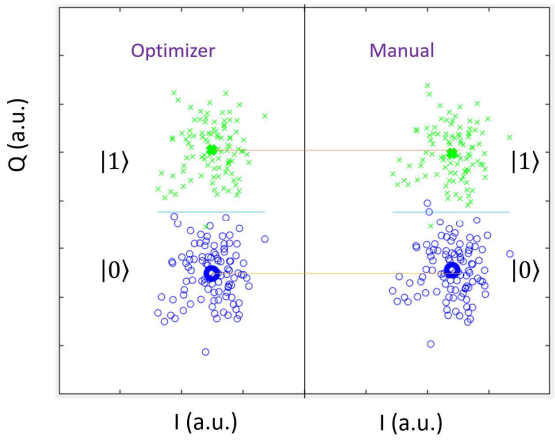

Figure 10: IQ-distribution plots of the inversely designed system by the proposed multi-objective optimization framework (Left) and by manual optimization (Right) for reading |0⟩ and |1⟩ states.

The solutions with a predicted readout error of 0 from Fig. 8 are then chosen for verification in the simulator. Fig. 9 shows the differences between the readout error predicted by ML/optimizer and simulation. The differences are larger than single objective optimization, particularly for the "Out-of-Range" case. However, it still can find a design with only 0.5% error with minimal P and t_p, which may reduce the disturbance to adjacent qubits and speed up the readout process. This point has the following parameters, χ = 122,932Hz, F = 7252534102Hz, P=-4.8dB, t_p=2.1μs, and Q=71603. Fig. 10 shows the IQ distribution of this optimal design simulated. It can be seen that the optimizer is able to provide a very optimal design as the two "blobs" just touch each other. Another simulation using the same P=-4.8dB, t_p=2.1μs but with other parameters using nominal values (which may be achieved manually) is also performed. Fig. 10 shows that it has an error of 1.5%. Moreover, the separation of the "blob" centroids is larger in the optimizer case.

V. Conclusion

A rapid inverse design and optimization framework has been demonstrated for superconducting qubit readout. It combines an analytical model, NN, and optimization algorithms. Both single-objective (differential evolution) and multi-objective optimization (NSGA-II) are performed. It can optimize a 5-dimensional parameter space within 3 days (including data generation) to find a design that is difficult to obtain manually.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. 2125906. The authors thank Yaniv Jacob Rosen and Kristin M. Beck for their support in the calibration process.

References

[1] D. DiVincenzo, "The Physical Implementation of Quantum Computation". Fortschritte der Physik. 48 (9–11): 771–783.

[2] F. Arute et al., 'Quantum supremacy using a programmable superconducting processor," Nature 574, 505–510 (2019).

[3] Hiu Yung Wong et al., "A Simulation Methodology for Superconducting Qubit Readout Fidelity," Solid-State Electronics, Volume 201, March 2023.

[4] Z. K. Minev et al., "Energy-participation quantization of Josephson circuits," npj Quantum Inf 7, 131 (2021).

[5] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," Proceedings of the 33rd International Conference on Neural Information Processing Systems, Article 721, pp. 8026-8037, 2019

[6] P. Virtanen et al., "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," Nature Methods, vol. 17, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.

[7] J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," in IEEE Access, vol. 8, pp. 89497-89509, 2020, doi: 10.1109/ACCESS.2020.2990567.

[8] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.