

# Superconducting Qubit Readout Pulse Optimization Using Deep Reinforcement Learning

by

Cole R. Hoffer

Submitted to the Department of Electrical Engineering and Computer  
Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author .....  
Department of Electrical Engineering and Computer Science  
February, 2021

Certified by .....  
William D. Oliver  
Associate Professor of Electrical Engineering and Computer Science  
Thesis Supervisor

Accepted by .....  
Katrina LaCurts  
Chair, Master of Engineering Thesis Committee



# Superconducting Qubit Readout Pulse Optimization

## Using Deep Reinforcement Learning

by

Cole R. Hoffer

Submitted to the Department of Electrical Engineering and Computer Science  
on February, 2021, in partial fulfillment of the  
requirements for the degree of  
Master of Engineering in Electrical Engineering and Computer Science

### Abstract

Quantum computers promise to solve specific problems significantly faster than classical computers. Superconducting quantum processors comprising more than 50 qubits can achieve quantum supremacy, the ability to outperform existing classical computers for particular problems. However, to build a useful quantum computer, the quantum processor's constituent components such as their control and readout must be very well-calibrated. Qubit-state readout of contemporary superconducting quantum processors is a significant error source. In an efficient, frequency-multiplexed readout of multiple qubits, effects such as drive cross-talk increase the complexity of optimal readout pulse shapes, requiring computationally intensive methods to discover high-fidelity pulse shapes. In this thesis, we extend existing readout optimization methods to work in multi-qubit environments and present a new pulse shaping optimization module using deep reinforcement learning. Compared to conventional readout methods in a simulated environment, we are able to reduce required readout pulse lengths by over 63% in single-qubit environments and by over 57% in multi-qubit environments. In addition to discussing how the deep reinforcement learning pulse shaping module will be used in experimental contexts, we also evaluate the future generalized use of deep reinforcement learning methods in quantum computing.

Thesis Supervisor: William D. Oliver

Title: Associate Professor of Electrical Engineering and Computer Science



## Acknowledgments

I want to start by thanking everyone in the EQuS group that supported me during my MEng. Importantly, I would like to thank my advisor, Professor Will Oliver, for taking a chance and providing me the opportunity to work in this fantastic group. The research was both challenging and fulfilling, and the knowledge and skills I gained this year will stick with me for a long time. Next, I would also like to send a huge thank you to Ben Lienhard, who served as a fantastic mentor when I first started at the group and became a great collaborator as the year went on. In no way would I have gotten as far as I did without all of his help and expertise. I would also like to thank Antti for being a great resource in helping scope this project and its experimental realization. Another shout out must go to Billy Woltz, one of my best friends at MIT and the one who helped me get the position in EQuS! Not only did he help me find this opportunity, but he was also a constant resource throughout my time here and was always a fabulous friend. Finally, I want to thank everyone else in the lab. While I was only on campus in the lab until March (thanks 2020!), everybody in the group was always supportive and helpful.

Next, I want to thank all those who have supported me this year and during my undergraduate years at MIT. First and foremost, my mom and dad are responsible for any success I've found here at MIT, and I am eternally grateful for their love and support. Second, I want to thank all of the close friends I've made at MIT. Their support, quirkiness, and late-night Domino's pizza breaks during my years at MIT were a major reason I was able to successfully pursue my academic goals. Finally, it's imperative that I especially thank Molly, whose support for me over the past thirteen years has been nothing short of amazing.



# Contents

<b>1</b>	<b>Introduction</b>	<b>19</b>
<b>2</b>	<b>Physics Background</b>	<b>23</b>
2.1	Quantum Computing Introduction . . . . .	23
2.2	Superconducting Quantum Computing . . . . .	24
2.2.1	Linear LC Circuits and Quantum Harmonic Oscillators . . . .	24
2.2.2	Josephson Junctions . . . . .	25
2.2.3	The Cooper Pair Box . . . . .	26
2.2.4	The Transmon Qubit . . . . .	28
<b>3</b>	<b>Superconducting Qubit Readout</b>	<b>31</b>
3.1	Cavity Quantum Electrodynamics . . . . .	32
3.2	Circuit QED and Dispersive Readout . . . . .	33
3.3	Multiplexed Readout . . . . .	36
3.4	Computational Simulation . . . . .	37
3.4.1	Cavity Bloch Equation Formulation . . . . .	37
3.4.2	Simulation System Parameters . . . . .	39
<b>4</b>	<b>Machine Learning Background</b>	<b>41</b>
4.1	Machine Learning . . . . .	41
4.2	Reinforcement Learning . . . . .	42
4.2.1	Markov Decision Processes . . . . .	44
4.2.2	Bellman Equation . . . . .	44

4.2.3	Value Iteration . . . . .	45
4.2.4	Policy Iteration . . . . .	46
4.3	Deep Learning and Neural Networks . . . . .	47
4.3.1	Feed-Forward Neural Networks . . . . .	47
4.3.2	Cost Functions and Regularization . . . . .	50
4.3.3	Training Neural Networks . . . . .	51
4.4	Deep Reinforcement Learning . . . . .	52
<b>5</b>	<b>Methodology</b>	<b>55</b>
5.1	Conventional Approach . . . . .	55
5.2	CLEAR Pulse . . . . .	56
5.2.1	Introduction . . . . .	56
5.2.2	Scaling to Multi-Qubit Systems . . . . .	56
5.2.3	Bayesian Optimization . . . . .	57
5.3	Deep Reinforcement Learning Generated Pulses . . . . .	60
5.3.1	Deep Q-Networks . . . . .	60
5.3.2	Proximal Policy Optimization . . . . .	63
5.3.3	Other Approaches Considered . . . . .	66
<b>6</b>	<b>Resonator Reset</b>	<b>69</b>
6.1	Introduction . . . . .	69
6.2	Optimization Setup . . . . .	70
6.3	Single Qubit Simulation . . . . .	71
6.3.1	Results . . . . .	71
6.3.2	Comparing DQN and PPO Optimizations . . . . .	73
6.4	5 Qubit Simulation . . . . .	75
6.4.1	Results . . . . .	75
6.4.2	Flexibility in Generated Pulse Shapes . . . . .	77
6.5	Summary . . . . .	77

<b>7</b>	<b>Resonator Photon Injection</b>	<b>79</b>
7.1	Introduction . . . . .	79
7.2	Optimization Setup . . . . .	80
7.3	Single Qubit Simulation Results . . . . .	82
7.4	5 Qubit Simulation Results . . . . .	84
7.4.1	Injection Results . . . . .	84
7.4.2	Entire Readout Pulse Results . . . . .	86
7.5	Summary . . . . .	88
<b>8</b>	<b>Experimental Setup</b>	<b>89</b>
8.1	Measurement Setup . . . . .	89
8.2	Labber Pulse Optimization Driver . . . . .	90
8.3	Resonator Reset Experiment Design . . . . .	91
8.4	Resonator Photon Injection Experiment Design . . . . .	91
<b>9</b>	<b>Conclusion and Future Work</b>	<b>95</b>
9.1	Conclusion . . . . .	95
9.2	Future Work Possibilities . . . . .	97
9.2.1	Additional Applications . . . . .	97
9.2.2	Pulse Length Optimization . . . . .	98
9.2.3	Continuous Experimental Optimization . . . . .	99



# List of Figures

2-1	Energy potential for a quantum harmonic oscillator (QHO) (a) and a transmon (b). In the QHO, all energy levels have equal spacing $\hbar\omega$ . In the transmon, the non-linear Josephson inductance (Eq. 2.11) creates non-equal spacing between each energy level, allowing the first two levels to be individually addressable and create the qubit's computational subspace. . . . .	25
2-2	Circuit diagrams for a Cooper pair box qubit (a) and a transmon qubit (b) where the crossed square is the symbol for the Josephson junction circuit element. The transmon builds upon the Cooper pair box architecture by including a large shunt capacitor to reduce sensitivity to charge noise. . . . .	27
2-3	The charge dispersion energies for the 3 first three qubit levels from Eq. 2.12. As the ratio $E_J/E_C$ increases, the levels exponentially flatten and enter the transmon regime. . . . .	28
3-1	General cartoon of single qubit dispersive readout. First, a readout resonator's frequency is shifted by the underlying qubit state. This shifted frequency can be probed by a readout pulse, whose reflected magnitude is effects based on that resonator frequency shift. These effects are articulated through the post processing of the readout signal output, where the qubit state can be determined by using discrimination techniques on the readout signal in the IQ-plane (b). . . . .	32

3-2	General representation of a cavity quantum electrodynamic system. Here a two level atom travels through the cavity in time $t_{\text{transit}}$ and decays at a rate $\gamma$ . While traveling through the cavity, the atom with the photons in the cavity with coupling strength $g$ and the photons in the cavity leak out of the cavity at a rate $\kappa$ . . . . .	34
3-3	(a) Circuit schematic of a 5 qubit multiplexed readout system, where each qubit is coupled to an individual resonator. The resonators are capacitively coupled to a common drive line. (b) The associated transmitted power for each qubit in the excited and ground states as a measure of the resonator frequency. . . . .	37
4-1	General flow of a Markov decision processes. Given a current state ( $s$ ) and reward ( $R$ ) at time step $t$ , an agent chooses an action that is fed into the environment which produces the next state and reward at for time step $t + 1$ , continuing the cycle until reaching a terminal state. .	44
4-2	(a) Basic feed-forward neural network architecture with an input layer of 3 nodes, 2 hidden layers with a width of 4 nodes each and single output layer node. (b) The value of each node in a neural network not in the input layer is determined by finding the sum of a bias term and the dot product between the previous layer's node values and the inter-layer weight values. This computed value is then fed into a non-linear activation function. . . . .	48
4-3	Common activation functions and their derivatives found in FNNs, including the linear, ReLU and sigmoid activation functions. . . . .	50
5-1	Arbitrary CLEAR and rectangular pulse shapes. Two amplitude segments at the beginning of the pulse help inject photons into the resonator quickly, allowing the resonator to reach a steady photon count in a shorter time. Two amplitude segments are also added to the end of the pulse to actively reset the resonator to the vacuum state. . . .	57

5-2	Wall-clock time needed per optimization step when optimizing different numbers of pulse segments. Regardless of the number of segments being optimized, the length of each optimization step generally increases as the optimization step number increases as we expect from the Bayesian optimization algorithm. Additionally, increasing the number of amplitude segments needing to be optimized also results in requiring more optimization time per step throughout the entire optimization process. . . . .	59
5-3	A simple Deep Q-Network architecture where we are trying to construct a single 4 segment pulse with 2 possible amplitude options. (a) A state is provided as the input for a deep fully-connected neural network. The neural network's outputs approximate the Q-value for each possible action according to the state input. (b) The action with the maximum associated Q-value is selected, appended to pulse, and sent to interact with the simulated or physical environment. (c) The state-action pair's actual Q-value is calculated in this environment and is saved in an experience replay buffer. (d) A batch of expected and actual Q-values with their state-action pairs are used to train the neural network. (e) The new state generated by the environment is passed back to the input of the neural network, restarting the process. . . . .	62
5-4	A simple Proximal Policy Optimization architecture where we are trying to build two 2-segment pulses where any bounded continuous amplitude can be chosen. We use two deep fully connected neural networks. The first is the policy network that aims to approximate an optimal policy (the best pulse shape). The second is the value network other that takes in the predicted pulse from the policy network and predicts the value of that pulse in the environment. . . . .	66

6-1	(a) Generated CLEAR, DQN and PPO pulse shapes for single qubit readout resonator reset. In this case, the conventional rectangular readout pulse passively waits for the photons to decay from the resonator, thus the pulse amplitude is 0. (b) Photon counts over time due to the generated pulse shapes. . . . .	72
6-2	2 dimensional representation of 25 dimensional pulses generated by DQN and PPO using t-SNE. With t-SNE, similar high dimensional data points are situated close together in the 2 dimensional representation. On the top row, each marker (pulse) is colored based on its reward, or number of photons remaining while on the bottom row each marker is colored based on the iteration it was made in its optimization cycle. Here we show every 50th pulse generated. . . . .	74
6-3	(a) Generated CLEAR, DQN, and PPO pulse shapes for single qubit readout resonator photon injection and stabilization. In this case, the conventional rectangular readout pulse passively waits for the photons to decay from the resonator, thus no pulse amplitude is applied. (b) Photon counts over time caused from the resulting generated pulse shapes. (c) Distribution of 25 generated CLEAR reset pulses. . . . .	76

7-1	A basic rectangular readout pulse at a normalized drive power $P_{\text{norm}} = 4$ (a) and the resulting photon counts for the resonators in a 5-qubit system (b). There are three main sections of the readout pulse that we consider. The first section is the photon injection optimization range at the beginning of the pulse, which we actively aim to minimize the required length of. Second, the middle section is a constant amplitude section that is driven at $\sqrt{P}$ and is equal in length to the preceding pulse section that is being length optimized. Finally the last segment is reset optimization section that was discussed in Chapter 6. As we seen in (b), the photon counts from a rectangular pulse have an oscillating evolution where the photon counts eventually stabilize based on the normalized drive power $P_{\text{norm}}$ . . . . .	80
7-2	(a) Generated CLEAR, DQN, and PPO pulse shapes for single qubit readout resonator photon injection and stabilization. In this case, the conventional approach is a single amplitude rectangular pulse. (b) Photon counts over time as a result of applying the generated pulse shapes. . . . .	83
7-3	Generated pulse shapes for photon injection and stabilization using a rectangular pulse, (a) CLEAR pulse and (b) PPO pulse. In this setup, the amplitudes of the first half of the pulse are optimized while the second half of the pulse is at a constant amplitude $\sqrt{P}$ . . . . .	85
7-4	Resonator photon counts for the full readout pulse (photon injection and reset) using rectangular pulses (left), optimized CLEAR pulses (center), and pulses generated using PPO (right). . . . .	86

7-5	IQ-plane cavity trajectories per resonator for each type of generated full readout pulse (rectangular, CLEAR, PPO). Trajectories with red markers are associated with a system where all 5 qubits are in the excited state, and vice versa for with ground state trajectories with blue markers. Markers are spaced 20ns apart to highlight the speed in which cavity states are reached. Each IQ-plane also has a dashed grey circle that represents a resonator state with 4 photons (stabilization target), and a smaller dotted grey circle that represents a resonator state with 0.10 photons (reset target).	87
8-1	(a) Simplified setup for qubit control and multiplexed readout for a superconducting multi qubit system. On the left, pulse shapes for qubit control and readout are created, combined, and sent to the 5-qubit chip. On the right, after passing through the chip, the readout signal returns to room temperature after passing through a chain of amplifiers, allowing us to acquire measurement data. (b) A magnified view of the 5-qubit chip in the super cooled mixing chamber. The chip's qubit's are connected to a single drive line in a multiplexed fashion, allowing us to control and readout all 5 qubits with a single multiplexed signal.	90
8-2	Sample Ramsey experiment pulse sequence. Here, a single Ramsey experiment begins by applying a set of $\pi$ -pulses to initialize the qubit states. This is followed by a rectangular readout pulse that has an optimized reset component at the end to empty the resonator as quickly as possible. Following the readout pulse, we apply two sets of $\pi/2$ -pulses, spaced apart by a time $t_R$ . These $\pi/2$ -pulses rotate the qubit-state dependent Bloch vector on the equator, and then back to the z-axis. During that time between the two $\pi/2$ -pulses, various amounts of detuning occurs which affects how vector returns back to it's original vector position following the second $\pi/2$ pulse.	92

# List of Tables

3.1	System parameters used in our simulations. Parameter values for the 5-qubit qubit chip based off a lab experiment focused on qubit-state discrimination. . . . .	39
9.1	Final readout pulse lengths for the various conventional and optimization methods in this research. The single-qubit environment includes optimized CLEAR pulses and pulses generated with PPO and DQN. In the multi-qubit environment, we only compare optimized CLEAR pulses and pulses generated with PPO. . . . .	97



# Chapter 1

## Introduction

Quantum computers promise to solve specific problems significantly faster than classical computers. Superconducting quantum processors comprising more than 50 qubits can achieve quantum supremacy, with the ability to outperform existing classical computers for particular problems [1, 2, 3]. To realize a useful quantum computer, however, the quantum processor’s constituent components, their control, and their readout must be very well-calibrated.

A critical component in any quantum computing system is the mechanism used to read out the system’s qubit states. For superconducting qubit systems, we look to perform quantum non-demolition measurements by using a method called dispersive readout, where each qubit is entangled with photons in a linear readout resonator [4]. The qubit shifts the resonator’s frequency in response to the qubit state, which allows us to readout the qubit state when that response is probed. This same technique scales to multiplexed readout, where we can control multiple qubit-resonator pairs simultaneously. However, scaling to multiplexed readout also introduces sources of cross-talk that can make optimizing ideal pulse shapes much more difficult, which we tackle here with deep reinforcement learning methods.

The goals of optimizing readout pulse shapes are to both quickly populate and stabilize the resonator states, as well as to actively reset the resonators as quickly as possible. First, quickly populating and stabilizing the resonator states allows us to shorten the time needed to probe the resonator’s dispersive shift. Although it is often

the case that having more fine-tuned pulse shape control allows us to shorten the required pulse length compared to a single segment rectangular pulse, the complexity of the optimal pulse shapes requires powerful optimization techniques. The second goal in optimizing readout pulse shapes is to actively reset the resonators following a measurement. The conventional method is to simply turn off the readout pulses and allow the photons to naturally decay from the resonator. This is a slow and passive process, which often requires waiting multiple cavity decay times and limits the performance of important tasks such as quantum error correction [5]. As with populating and stabilizing the resonator with photons, we can actively shorten the time required to reset the resonator states using an optimized, fine-tuned pulse shape.

General improvements have been realized by engineering readout pulse shapes that populate and stabilize the readout resonator as well as actively reset it. A currently used example is the Cavity Level Excitation and Reset (CLEAR) pulse, a rectangular pulse that includes constant amplitude pulses before and after the rectangular pulse to quicken this excitation and reset process [6]. The CLEAR pulse method actively depopulated the resonator around two cavity time constants faster than the conventional approach. While the CLEAR pulse helps quicken the readout process, it also raises the question of how the CLEAR pulse can be scaled to multiplexed readout systems, as well as if more fine-tuned control of the readout pulses can further reduce the required readout pulse lengths.

These questions guide the focus of this thesis, which applies deep reinforcement techniques for fine-tuned optimal control of readout pulse shapes. Reinforcement learning is a subset of machine learning where an agent learns to maximize a reward by choosing the appropriate action at a given state [7]. Rather than comparing predictions with labeled data sets like in supervised learning applications (e.g., image classification), reinforcement learning algorithms “learn” through the process of their agents directly interacting with the environment. Additionally, reinforcement learning algorithms do not need to know how the underlying system functions. Rather, they simply need to be able to compute the reward that a specific state-action pair will produce. The growth of reinforcement learning research has been shown through

success in complex games such as chess and Go, as well as robot planning [8, 9, 10].

Optimization environments with a relatively small number of state-action pairs like Tic-Tac-Toe can be solved through a variety of tabular methods where we attempt to learn the expected reward for all state-action pairs [7]. However, while tabular reinforcement learning techniques provide a strong foundation for problems like optimal control, those techniques quickly become impractical with increased dimensionality and precision. Tabular solution methods require us to approximate rewards for every state-action pair in the environment, a number that grows exponentially as the state or action set increases. Approximate solution methods address this issue by replacing the state-action pair table with a deep neural network that approximates the expected reward for executing an action at a particular state. These new deep reinforcement algorithms have greatly expanded the size and complexity of problems that can be addressed with reinforcement learning. These algorithms have begun to be used on problems in quantum optimal control, such as unitarily steering a quantum system towards a desired dynamic [11].

Critical to the overall success of this research is having the resources to explore and tune different reinforcement algorithms on actual quantum processors. However, we do not have unlimited access to our laboratory’s quantum processors and thus need to initially rely on robust simulation software. The implementation realities of deep reinforcement learning add to the necessity of beginning the work with a simulator. First, deep reinforcement learning algorithms are notoriously resource-intensive to train and tune. Initially relying on a simulator allows us to take advantage of resources like cloud computing and specialized GPUs that immensely improve training speeds and algorithm iterations. Second, training these deep reinforcement algorithms on the simulator provides a training “head start” before beginning work on the actual quantum computer. We will constrain our simulator to the limitations of the physical devices we will be using as much as possible (e.g., limiting the set of pulse amplitudes provided by the arbitrary waveform generator). While the simulator is not able to account for all effects that may be experienced by the lab’s quantum computer, it will allow the algorithms to train purely on the base-level readout physics. Thus, the

majority of training on the quantum computer will be fine-tuning, saving a significant amount of training time.

For the structure of this thesis, we will start with an introduction to the building blocks of superconducting qubits in Chapter 2. In Chapter 3, we will discuss the basics of cavity QED and its counterpart circuit QED , which will also allow us to introduce the physics behind superconducting qubit readout. From there, we will transition to an introduction to machine learning and reinforcement learning in Chapter 4. We can then go into depth about the various deep reinforcement learning algorithms we use to optimize optimal multiplexed readout pulse shapes in Chapter 5. These algorithms will drive our simulation results both with optimizing pulses that populate and stabilize readout resonators to reduce the time needed to perform qubit measurements (Chapter 7), as well as optimizing pulses that actively reset those resonators following a measurement (Chapter 6). Finally in Chapter 8, we will describe the experimental designs that are being conducted based off the previous simulation results.

# Chapter 2

## Physics Background

### 2.1 Quantum Computing Introduction

While a classical bit is represented as a binary digit with values 0 and 1, a qubit can be in a superposition of both 0 and 1. We write the quantum state of a single qubit as a wave function

$$|\Psi\rangle = \alpha |0\rangle + \beta |1\rangle, \quad (2.1)$$

where  $|\alpha|^2 + |\beta|^2 = 1$ . If our quantum system has  $N$  qubits, there will be a resulting complex superposition of all  $2^N$  permutations, where the system's entire state can be defined by the wave function

$$|\Psi\rangle = c_1 |0..00\rangle + c_2 |0..01\rangle + \dots + c_{2^N} |1..11\rangle. \quad (2.2)$$

Due to the ability to store exponentially more information than its classical counterpart, and to apply quantum transformations on all states simultaneously, quantum processors can tackle many difficult problems that their classical counterparts cannot. These include, but are not limited to, factoring large numbers exponentially quicker using Shor's algorithm, as well as simulating complex quantum systems that can be used in applications like chemistry and physics [12, 13, 14]. In this thesis, we focus on superconducting qubits as the foundation for building these quantum processors.

## 2.2 Superconducting Quantum Computing

### 2.2.1 Linear LC Circuits and Quantum Harmonic Oscillators

To understand superconducting qubits, we can start from the basics with a classical linear  $LC$  resonant circuit. This  $LC$  circuit serves as an electrical counterpart to a simple mechanical harmonic oscillator. We first start by defining  $\Phi$  as the branch flux on inductor  $L$  and by defining  $Q$  as the branch charge on the capacitor crossing voltage  $V$ . This leads to the energy terms of the capacitor and inductor respectively being defined as

$$\mathcal{T}_C = \frac{Q^2}{2C}, \quad (2.3)$$

$$\mathcal{U}_L = \frac{\Phi^2}{2L}. \quad (2.4)$$

Using the Legendre transformation (the difference between the kinetic and potential energy terms), the *classical* Hamiltonian is further defined as

$$H = \frac{Q^2}{2C} + \frac{\Phi^2}{2L}. \quad (2.5)$$

In order to enter the quantum regime, we use the quantum operators  $\hat{Q}$  and  $\hat{\Phi}$  to replace  $Q$  and  $\Phi$ , where  $[\hat{\Phi}, \hat{Q}] = \hat{\Phi}\hat{Q} - \hat{Q}\hat{\Phi} = i\hbar$ . Naturally, this leads to the resulting quantum Hamiltonian being defined as

$$H = \frac{\hat{Q}^2}{2C} + \frac{\hat{\Phi}^2}{2L}. \quad (2.6)$$

This Hamiltonian is equivalent to the Hamiltonian of a quantum harmonic oscillator (QHO), which describes a particle in a one-dimensional quadratic potential. We define  $\hbar$  as the reduced Planck's constant where  $\hbar = h/2\pi$ ,  $\phi$  as the reduced flux or phase across an inductor, and finally  $a$  as the annihilation operator and  $a^\dagger$  as the creation operator. Setting  $\omega_r = 1/\sqrt{LC}$  as the systems resonant frequency, the QHO

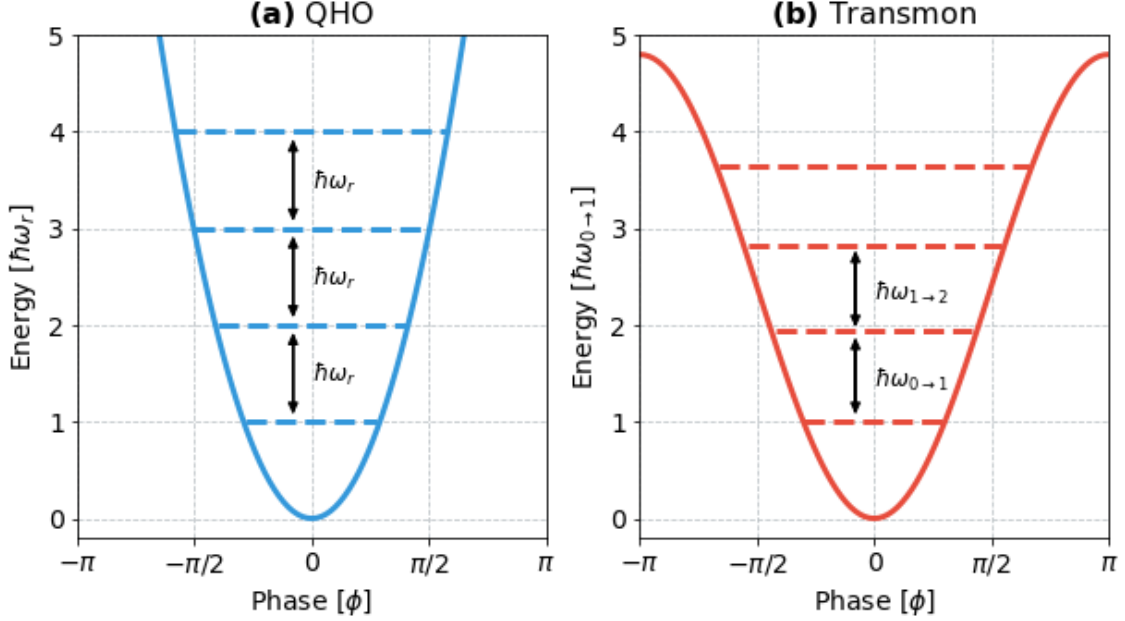


Figure 2-1: Energy potential for a quantum harmonic oscillator (QHO) (a) and a transmon (b). In the QHO, all energy levels have equal spacing  $\hbar\omega$ . In the transmon, the non-linear Josephson inductance (Eq. 2.11) creates non-equal spacing between each energy level, allowing the first two levels to be individually addressable and create the qubit's computational subspace.

Hamiltonian in the second quantization form is defined as

$$H = \hbar\omega_r \left( a^\dagger a + \frac{1}{2} \right). \quad (2.7)$$

A major limitation of quantum harmonic oscillators in quantum computing is that energy levels are equally spaced at  $\hbar\omega_r$  (Fig. 2-1). To build a qubit, we need to be able to address two of these energy states exclusively to build a computational subspace. More specifically, we need  $\omega_{0 \rightarrow 1}$  to differ from  $\omega_{1 \rightarrow 2}$  to address these states as part of computational subspace. The work to address this issue is introduced in the forthcoming sections.

### 2.2.2 Josephson Junctions

To fix the problem of having quantized energy levels being evenly spaced, we introduce a non-linearity, or anharmonicity, into our system using Josephson junctions. A

Josephson junction is formed between two superconducting electrodes that are separated by an insulating barrier, in which Cooper pairs can coherently tunnel across that insulating barrier [15]. Two relations define the dynamics of Josephson junctions. The first Josephson relation defines the current flowing through the junction as

$$I = I_c \sin(\phi(t)), \quad (2.8)$$

where  $I_c$  is defined as the critical current of the junction (the maximum supported current that can be sustained by the junction) and  $\phi(t)$  is defined as the phase difference across the junction. That phase difference evolves in the presence of a potential  $V$  such that

$$V = \frac{\hbar}{2e} \frac{d\phi}{dt}. \quad (2.9)$$

As  $\Phi_0 = \frac{\hbar}{2e}$  is the superconducting magnetic flux quantum, we can use the two relations to see that

$$V = \frac{\Phi_0}{I_c \cos(\phi)} \frac{dI}{dt}. \quad (2.10)$$

Combined with Faraday's Law  $V = -L \frac{dI}{dt}$ , this allows us to define the non-linear Josephson inductance as

$$L_J = \frac{\Phi_0}{2\pi I_c \cos(\phi)}. \quad (2.11)$$

Having the inductance be non-linear results in a non-linear oscillator, thus allowing the Josephson junction (and combination of junctions) to serve as the backbone of building superconducting qubits. We will now introduce two of those qubits, the Cooper pair box, and the proceeding transmon qubit.

### 2.2.3 The Cooper Pair Box

One of the simplest superconducting qubits is the Cooper pair box (or charge qubit), which creates a superconducting island that is connected to a superconducting reservoir using the previously described Josephson junction (Fig. 2-2) [16]. The Cooper pair box qubit's states are dependent on the presence of additional Cooper pairs in the superconducting island.

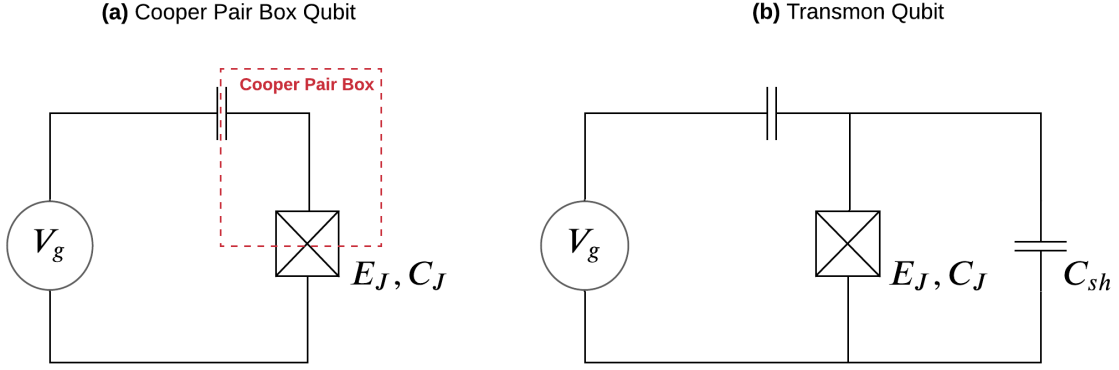


Figure 2-2: Circuit diagrams for a Cooper pair box qubit (a) and a transmon qubit (b) where the crossed square is the symbol for the Josephson junction circuit element. The transmon builds upon the Cooper pair box architecture by including a large shunt capacitor to reduce sensitivity to charge noise.

The Cooper pair box Hamiltonian is defined as

$$H = 4E_C(\hat{n} - n_g)^2 - E_J \cos \phi, \quad (2.12)$$

where  $\hat{n}$  is the Cooper pair number operator representing the number of excess Cooper pairs on one of the islands,  $n_g$  is the effective offset charge, and the reduced flux  $\phi$  is a conjugate pair to  $\hat{n}$  [17]. Here,  $E_J$  is the Josephson energy and  $E_C = e^2/2C_\Sigma$  where  $C_\Sigma = C_S + C_J$  is the total capacitance including both the shunt capacitance and the capacitance of the junction. We can also write the Hamiltonian in the charge basis as

$$H = 4E_C(\hat{n} - n_g)^2 - \frac{E_J}{2} \sum_n (|n\rangle \langle n+1| + |n+1\rangle \langle n|). \quad (2.13)$$

The Cooper pair box is operated in a regime where  $E_J \ll 4E_C$  so the eigenstates are number-like. A major issue with the Cooper pair box operating as a qubit is its sensitivity to  $1/f$  charge noise [18]. One mitigation approach is to operate the system at “sweet spots” where the charge dispersion slope is zero at  $n_g \pm 0.5$  [19]. Operating at the sweet spots allows one to reduce the system to a two-level qubit, as states with the  $|n\rangle$  and  $|n+1\rangle$  Cooper pairs form avoided level crossings (Fig. 2-3). These are also the points with the largest anharmonicity, making them advantageous for

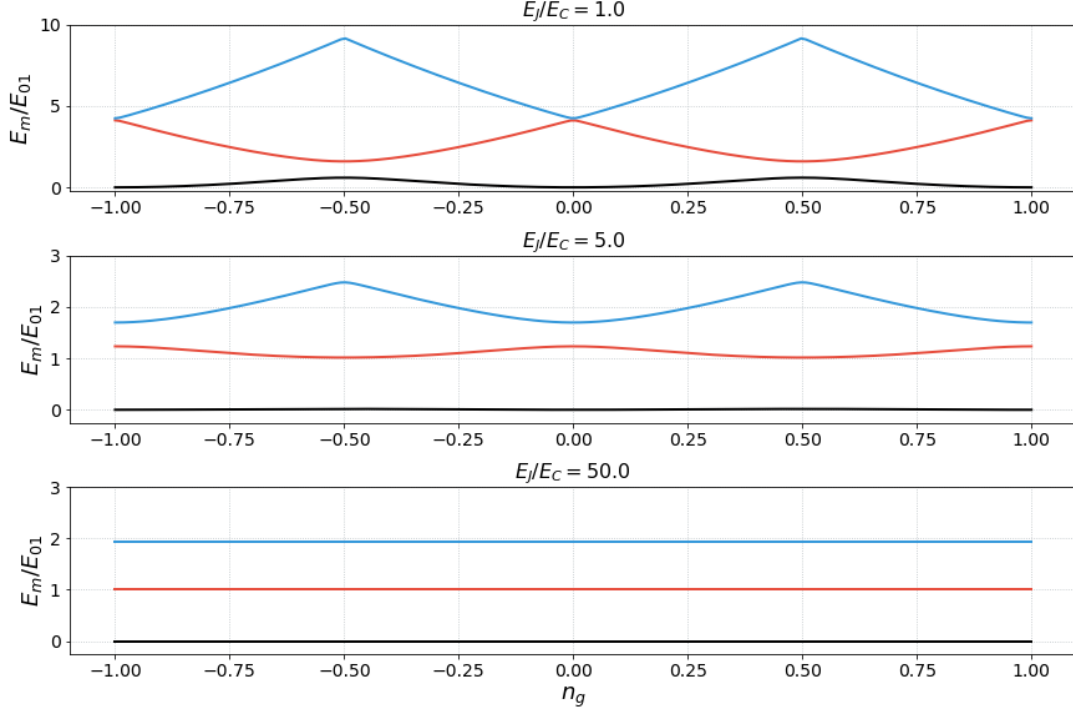


Figure 2-3: The charge dispersion energies for the 3 first three qubit levels from Eq. 2.12. As the ratio  $E_J/E_C$  increases, the levels exponentially flatten and enter the transmon regime.

assigning the ground and first excited states as the computational subspace  $|0\rangle$  and  $|1\rangle$ . Unfortunately, while this approach can help raise the coherence times significantly, the location(s) of the sweet spot can change frequently due to system fluctuations, requiring the gate voltage to be reset often.

## 2.2.4 The Transmon Qubit

The transmon qubit extends the Cooper pair box by including a large shunt capacitor that allows the qubit to reduce its sensitivity to charge noise (Fig. 2-2) [20]. While the Hamiltonian describing the transmon is equal to that of the Cooper pair box (Eq. 2.12), the addition of a large shunt capacitor allows the transmon to operate in a regime where the  $E_J/E_C$  ratio significantly exceeds that of a Cooper pair box. By increasing that ratio ( $E_J/E_C = 50$ ), the charge dispersion decreases exponentially, leading to a qubit transition frequency that is very insensitive to charge noise as depicted in Fig. 2-3(c).

The transmon frequency is that of an anharmonic LC oscillator, which can be calculated similarly to the QHO except that the transition frequency is now

$$\omega_q = \frac{1}{\sqrt{L_J C_\Sigma}} \approx \sqrt{8E_J E_C}. \quad (2.14)$$

We often want to be able to adjust the frequency of a transmon, which we can accomplish by pairing junctions in parallel in what is known as a DC SQUID (superconducting quantum interference device). By applying an external magnetic flux  $\Phi_{\text{ext}}$  through the SQUID, we can tune the effective critical current and thus the qubit's Josephson energy. Having these junctions be symmetric introduces flux noise that can affect the transmon qubit frequencies, thus we design these SQUID loops to be asymmetric [21]. Looking at our original transmon Hamiltonian in Eq. 2.12, we update our Josephson energy term  $E_J$  such that

$$E'_J = E_{J\Sigma} \sqrt{\cos^2 \left( \frac{\pi \Phi_{\text{ext}}}{\Phi_0} \right) + d^2 \sin^2 \left( \frac{\pi \Phi_{\text{ext}}}{\Phi_0} \right)}, \quad (2.15)$$

where  $E_{J\Sigma} = E_{J1} + E_{J2}$ , we define  $d$  as the junction asymmetry parameter  $d = ((E_{J2}/E_{J1} - 1)/((E_{J2}/E_{J1} + 1))$ . The resulting transmon Hamiltonian is now

$$H = 4E_C(\hat{n} - n_g)^2 - E'_J \cos \phi, \quad (2.16)$$

and the transmon qubit frequency is now tunable with respect to  $\Phi_{\text{ext}}$

$$\omega_q(\Phi_{\text{ext}}) = \sqrt{8E'_J E_C}. \quad (2.17)$$

Altogether, while the anharmonicity (the difference between the 0<sup>th</sup> and 1<sup>st</sup> state separation and the 1<sup>st</sup> and 2nd state separation) is reduced in the Transmon, we can now specifically address the two lowest levels as a qubit (Fig. 2-1). Additionally, the asymmetric transmon is both tunable and resistant to charge noise allowing for higher coherences throughout the system.



# Chapter 3

## Superconducting Qubit Readout

One of the critical components of any quantum computing system is the ability to perform fast and high fidelity qubit state readout. We specifically look at quantum non-demolition readout (QND), where the outcome of the measurement is not affected by the readout process. The general readout process is illustrated in Fig. 3-1, where applying a rectangular pulse at the resonator frequency allows us to probe the qubit state-dependent dispersive shift in the resonator. That dispersive shift is picked up by the readout signal after it interacts with the resonator and following of post processing effort, that signal data can be post-processed and used to discriminate qubit states in what is known as the IQ-plane. Here, the IQ-plane refers to the real and an imaginary part, or an “in-phase” (I) and a “quadrature” (Q) part of a modulating signal that changes the carrier’s frequency slightly.

The specific goal of this research is to optimize the control pulse shapes of the readout process in terms of populating the readout resonators more quickly to probe that dispersive shift and actively resetting the resonators so future operations are not affected and can be performed efficiently. The actual post-processing of the readout signal and subsequent discrimination of qubit states is outside the scope of this work as it is another active area of research in improving readout [22].

We will begin this chapter by first covering how individual atoms interact with photons in a cavity through cavity quantum electrodynamics (QED). We then transition to circuit QED, an analog of cavity QED where we now use a microwave-frequency

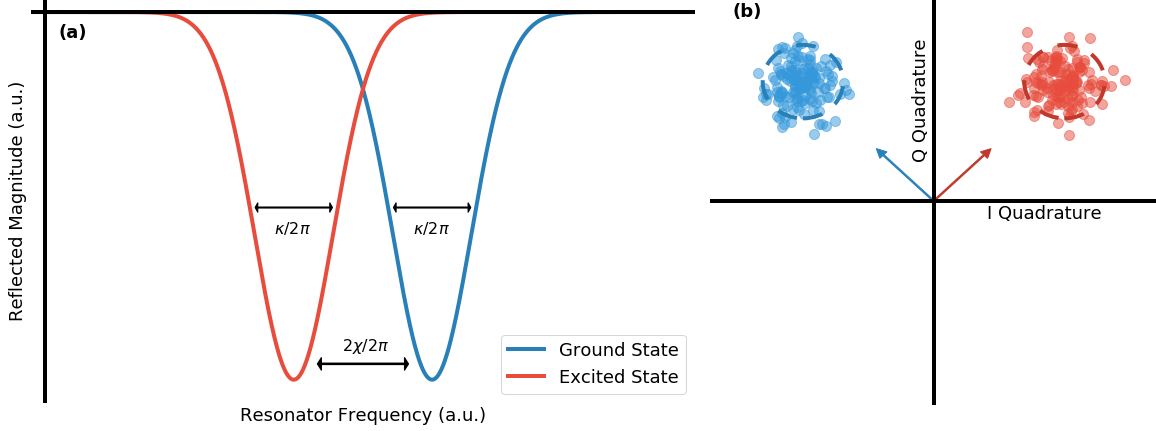


Figure 3-1: General cartoon of single qubit dispersive readout. First, a readout resonator's frequency is shifted by the underlying qubit state. This shifted frequency can be probed by a readout pulse, whose reflected magnitude is effects based on that resonator frequency shift. These effects are articulated through the post processing of the readout signal output, where the qubit state can be determined by using discrimination techniques on the readout signal in the IQ-plane (b).

waveguide transmission line resonator coupled to superconducting qubits that allow us to perform quantum non-demolition readout. Finally, we scale from single-qubit readout to multiplexed dispersive readout of multiple qubits.

### 3.1 Cavity Quantum Electrodynamics

In cavity quantum electrodynamics (QED), we look at the interactions between atoms and harmonic oscillator excitations (specifically focusing on microwave photons here). As shown in Fig. 3-2, by placing a two-level atom in a cavity, the coherent interactions between the atom and photons passing through the atoms can be assessed using the photons leaking from the cavity. This system can be described by the Jaynes-Cummings Hamiltonian

$$H_{JC} = \omega_r \left( a^\dagger a + \frac{1}{2} \right) + \frac{\omega_a}{2} \sigma_z + g(a\sigma_+ + a^\dagger\sigma_-), \quad (3.1)$$

where  $\omega_a$  is the atomic transition frequency,  $\omega_r$  is the cavity/resonator frequency, and  $\sigma_z$  is the Pauli-z operator. The final term  $g(a\sigma_+ + a^\dagger\sigma_-)$  describes the dipole interaction of the atom absorbing a photon from or emitting a photon to the cavity

at the atom-cavity coupling strength  $g$ , where  $\sigma_+ = |0\rangle\langle 1|$  and  $\sigma_- = |1\rangle\langle 0|$ .

In cavity QED, we work in what is known as the strong coupling regime, where the atom-cavity coupling strength  $g \gg \kappa$  and  $g \gg \gamma$ , where  $\kappa$  and  $\gamma$  are the cavities and atom decay rates. In the strong coupling regime, if the cavity and atom are at the same frequency the states  $|0, \uparrow\rangle$  (cavity in the ground state, atom in an excited state) and  $|1, \downarrow\rangle$  (cavity in an excited state, atom in a ground state) oscillate at the frequency  $g/2\pi$ . Being in the  $n$ -excitation state, the energy difference between the states is  $2g\sqrt{n}$  which creates an anharmonicity ( $\delta$ ) and thus the ability to function as a qubit.

We also often aim to operate cavity QED in what is known as the dispersive regime, where the atom and resonator detuning is large such that  $|\Delta| = |\omega_a - \omega_r| \gg g$ . By expanding to the second-order in  $g$  and making the unitary transformation

$$U = \exp\left\{\frac{g}{\Delta}(a\sigma_+ + a^\dagger\sigma_-)\right\}, \quad (3.2)$$

the updated Jaynes-Cummings Hamiltonian in the dispersive regime can be approximated as

$$H_{\text{JC,disp}} \approx UH_{\text{JC}}U^\dagger = \left(\omega_r + \frac{g^2}{\Delta}\sigma_z\right)\left(a^\dagger a + \frac{1}{2}\right) + \frac{\omega_a}{2}\sigma_z. \quad (3.3)$$

Here, the cavity frequency is dispersively shifted by  $\frac{g^2}{\Delta}$  depending on the atom state  $\sigma_z$ . Here, instead of using the Pauli-z matrix, we can just approximate  $\sigma_z$  as being  $\sigma_z = -1$  if the atom is in the excited state or  $\sigma_z = 1$  if the atom is in the ground state.

## 3.2 Circuit QED and Dispersive Readout

The cavity QED formulation can also be applied to superconducting circuits in what is known as circuit QED [23, 24]. The “cavity” in circuit QED is now a microwave-frequency coplanar transmission line resonator. Superconducting qubits are placed at a voltage anti-node in the resonator with qubit-resonator interaction strength  $g$ . We again operate in both the strong coupling regime and the dispersive regime as

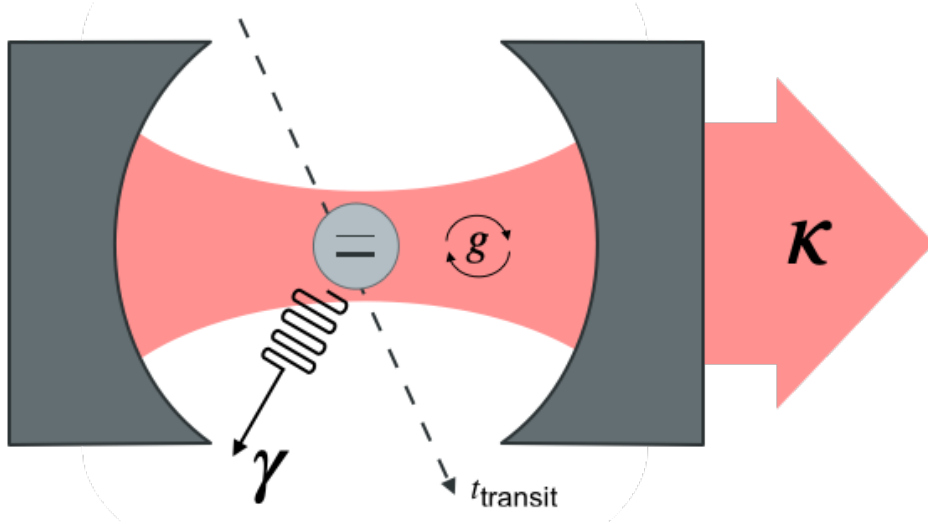


Figure 3-2: General representation of a cavity quantum electrodynamic system. Here a two level atom travels through the cavity in time  $t_{\text{transit}}$  and decays at a rate  $\gamma$ . While traveling through the cavity, the atom with the photons in the cavity with coupling strength  $g$  and the photons in the cavity leak out of the cavity at a rate  $\kappa$ .

the qubit state-dependent frequency shift can be probed to perform QND readout. The modified Jaynes-Cummings Hamiltonian operating in the dispersive regime for circuit QED is defined as

$$H_{\text{JC,disp}} \approx \omega_r a^\dagger a + \left( \omega_q + 2\chi \left[ a^\dagger a + \frac{1}{2} \right] \right) \frac{\sigma_z}{2} \quad (3.4)$$

where the dispersive shift is represented by  $\chi = g^2/\Delta$  and the qubit transition frequency is  $\omega_q$  [25]. Additionally, to include the effect of introducing a drive of frequency  $\omega_d$  on the resonator, we define a drive Hamiltonian

$$H_D = \epsilon(t)(a^\dagger e^{-i\omega_d t} + a e^{i\omega_d t}), \quad (3.5)$$

where  $\epsilon(t)$  is the drive amplitude at time  $t$  and  $\omega_d$  is the drive frequency [26]. To choose appropriate drives, we can set  $\omega_d = \omega_r$ , which allows for the average number of photons in a drive to be independent of the underlying qubit state in the absence of any non-linear corrections. One such non-linearity we include in our simulator is the Kerr non-linearity, a behavior that negatively shifts the cavity frequency from the measurement frequency when in the ground state (and vice-versa for the excited

state). The Kerr non-linearity and its subsequent Hamiltonian are defined as

$$K = \frac{2g^4\delta(3\omega_q^4 + 2\omega_q^2\omega_r^2 + 3\omega_r^4)}{(\omega_q^2 - \omega_r^2)^4}, \quad (3.6)$$

$$H_K = K(a^\dagger a)^2. \quad (3.7)$$

Finally, to incorporate cavity dampening as a function of the cavity photon decay rate  $\kappa$ , we can combine our previously defined Hamiltonians and use the master equation

$$H = H_{\text{JC, disp}} + H_D + H_K \quad (3.8)$$

$$\dot{\rho} = -i[H, \rho] + \kappa\hat{\mathcal{D}}[a]\rho. \quad (3.9)$$

Based on these calculations and parameters, there are a couple of important values we can keep track of throughout our work. First, we can define the critical photon number as

$$n_{\text{crit}} = \frac{\Delta^2}{4g^2}. \quad (3.10)$$

The critical photon number a threshold. Exceeding that quantity of photons in a resonator will result in the dispersive Hamiltonian defined in Eq. 3.4 no longer be a valid approximation. For our simulator, we use these quantities as an upper bound for how many photons can populate a resonator when generating optimized readout pulses. Another important parameter we can calculate is the  $P_{\text{1ph}}$ , the steady drive power required to inject one photon into the cavity at a steady state. We can get a close approximation of  $P_{\text{1ph}}$  by finding the steady-state solution of  $H_{\text{JC, disp}} + H_D$ ,

$$1 = a^\dagger a = \frac{P_{\text{1ph}}}{\chi^2 + (\kappa/2)^2}. \quad (3.11)$$

Once we have  $P_{\text{1ph}}$ , we can drive the resonator for a sufficiently long time in order to inject  $P_{\text{norm}}$  photons, where the drive amplitude  $\epsilon = \sqrt{P_{\text{norm}}P_{\text{1ph}}}$  and  $P_{\text{norm}}$  is the ratio between the applied power and  $P_{\text{1ph}}$ .

### 3.3 Multiplexed Readout

One of the key challenges for scaling up superconducting quantum processors to hundreds or thousands of qubits is how to scale the control and readout infrastructure as the number of qubits increases. It is infeasible to have individual amplification chains for every qubit as we develop chips high number of qubits. Instead, we look to implement a frequency-multiplexed design, where multiple qubit-resonator pairs are controlled on a single amplification chain. To individually control qubits, we space their respective resonator operating frequencies, often in the range of 50-100 MHz. The more qubits we want on a single drive line, the closer together those frequency spacing's are going to be, amplifying the need for optimized pulse shapes that can handle various forms of cross-talk.

The important next question is how do we control these multiplexed qubits simultaneously? This can be accomplished by using what is known as frequency-multiplexing. For a single pulse, separate real and imaginary parts of a complex signal,  $A_I(t)$  and  $A_Q(t)$ , are inputted into an IQ-mixer with local oscillator frequency  $\omega_{LO}$ . The output of the mixer and our readout signal is

$$A_{in} = A_I(t) \cos(\omega_{LO}t) - A_Q(t) \sin(\omega_{LO}t) = \text{Re}\{(A_I(t) + iA_Q(t))e^{i\omega_{LO}t}\}. \quad (3.12)$$

To apply multiple readout tones simultaneously, we simply calculate the overall readout signal as the sum of readout tones that are set at different frequencies. Our frequency-multiplexed readout signal is now defined as

$$A_{in} = \sum_j \text{Re}\{A_j(t)e^{i\omega_{RO,j}t}\}, \quad (3.13)$$

where  $A_j = A_{I,j} - iA_{Q,j}$  and  $\omega_{d,j}$  is the drive frequency for qubit  $j$  such that  $\omega_{RO,j} = \omega_{LO} + \omega_{d,j}$ .

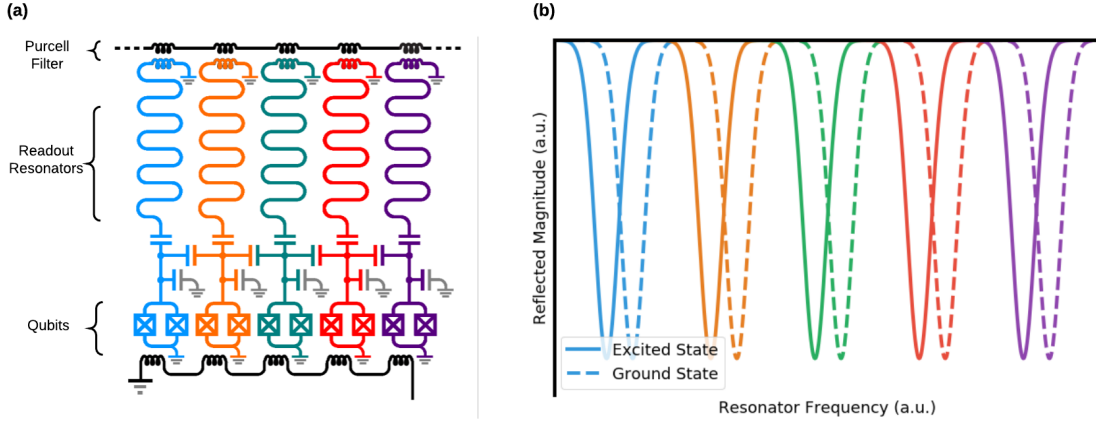


Figure 3-3: (a) Circuit schematic of a 5 qubit multiplexed readout system, where each qubit is coupled to an individual resonator. The resonators are capacitively coupled to a common drive line. (b) The associated transmitted power for each qubit in the excited and ground states as a measure of the resonator frequency.

### 3.4 Computational Simulation

Because this research is focused on exploring machine learning approaches to optimize pulse shapes, it's important to have robust simulation software that allows us to test and iterate on many models and training hyper-parameter sets. In the previous sections, we described a single qubit dispersive readout through the master equation. Master equations such as in Eq. 3.9 can be solved with existing software libraries like QuTip [27]. However, these master equation solvers are particularly slow, often taking up to a minute of wall-clock time just to simulate a simple rectangular readout pulse. Obviously, this is way too slow for machine learning algorithms that needs many simulated pulses to find optimal pulse shapes, so we look to convert the master to a Bloch equation formulation.

#### 3.4.1 Cavity Bloch Equation Formulation

The cavity Bloch equation formulation allows us to describe the evolutionary behavior of the system in terms of change over time [28]. Specifically, we can focus on the change over time for specific operators, like the photon count operator  $a^\dagger a$  and the cavity state operator  $a$ . Even more importantly, we found that they offer an impor-

tant computational speedup compared to existing master equation and Monte-Carlo solvers in software libraries like QuTiP, enabling us to run more accurate and more frequent simulations [27].

We first begin with defining the cavity Bloch equation for the evolution of an arbitrary operator  $A$  as

$$d_t \langle A \rangle = \text{Tr}\{A\dot{\rho}\} = -i\langle [A, H]\rho \rangle + \kappa \langle \hat{\mathcal{D}}[a]A \rangle. \quad (3.14)$$

Additionally, the dissipator term can be rewritten as

$$\hat{\mathcal{D}}[a]A = \frac{1}{2}(2a^\dagger Aa - a^\dagger aA - Aa^\dagger a). \quad (3.15)$$

Now with the general formulation we can find the cavity Bloch equation for the evolution of the cavity state operator  $a$  and photon count operator  $a^\dagger a$  using the Hamiltonian from Eq. 3.8. For the cavity state operator  $a$ ,

$$\begin{aligned} d_t \langle a \rangle &= -i\langle [a, H]\rho \rangle + \kappa \langle \hat{\mathcal{D}}[a]a \rangle \\ &= -i\Delta_{\text{rm}} - i\chi \langle a\sigma_z \rangle + i\epsilon(t) - iK \langle a \rangle - iK \langle a^\dagger aa \rangle - \frac{\kappa}{2} \langle a \rangle. \end{aligned} \quad (3.16)$$

Here  $\Delta_{\text{rm}} = |\omega_r - \omega_d|$ . In a single qubit system with only one pulse,  $\Delta_{\text{rm}} = 0$  as we drive the pulse at the resonator frequency. For the photon count operator  $a^\dagger a$  we find the cavity Bloch equation to be,

$$\begin{aligned} d_t \langle a^\dagger a \rangle &= -i\langle [a^\dagger a, H]\rho \rangle + \kappa \langle \hat{\mathcal{D}}[a]a^\dagger a \rangle \\ &= -2\epsilon(t) \text{Im}\{\langle a \rangle\} - \kappa \langle a^\dagger a \rangle. \end{aligned} \quad (3.17)$$

Next, we look to extend these formulations to multiplexed readout environments. This extension to having multiple pulses on the same drive leads to various forms of cross-talk. The most significantly impacting source of cross-talk the control or readout pulse for one qubit-resonator pair having an impact on neighboring qubit-resonator pairs. Additionally, we also look at coupling effects between neighboring qubits and resonators. For the cavity state operator  $a$  and photon number operator

$a^\dagger a$ , we update the cavity Bloch equations for each resonator  $i$  to be defined as

$$d_t \langle (a)_i \rangle = -i \sum_{j \in R} (\Delta_{\text{rm}(i,j)} - \epsilon_j(t)) - i \sum_{j \in Q} (\chi_{(i,j)} \langle a_i \sigma_{z(j)} \rangle + K_{(i,j)} \langle a_i \rangle + K_{(i,j)} \langle (a^\dagger a)_i a_i \rangle) - \frac{\kappa_i}{2} \langle a_i \rangle, \quad (3.18)$$

$$d_t \langle (a^\dagger a)_i \rangle = -2\epsilon_i(t) \text{Im}\{\langle a_i \rangle\} - \kappa \langle (a^\dagger a)_i \rangle. \quad (3.19)$$

Here, the effects of control cross-talk are encapsulated in  $-i \sum_{j \in R} (\Delta_{\text{rm}(i,j)} - \epsilon_j(t))$ , as the absolute difference between the drive and resonator frequencies will not be equal to zero for drives targeting other qubit-resonator pairs. Additionally, the effects of the qubit  $j$  interacting with resonator  $i$  are picked up in the dispersive shifts and Kerr non-linearities  $-i \sum_{j \in Q} (\chi_{(i,j)} \langle a_i \sigma_{z(j)} \rangle + K_{(i,j)} \langle a_i \rangle + K_{(i,j)} \langle (a^\dagger a)_i a_i \rangle)$ . For practical purposes in our simulations, we reduce the set of cross-talk interactions only to nearest neighbors, as beyond that the cross-talk contributions become negligible.

### 3.4.2 Simulation System Parameters

For our computational simulations, we use the 5-qubit chip system parameters from a recent lab experiment using the same chip we will perform our own real experiments on, for which the data can be found in Tab. 3.1 below [22].

	Qubit					Resonator					
	$\omega_q/2\pi$ (GHz)	$T_1$ ( $\mu s$ )	$T_2$ ( $\mu s$ )	$T_{2E}$ ( $\mu s$ )	$\omega_r/2\pi$ (GHz)	$\kappa/2\pi$ (MHz)	$\chi/\kappa$	$g/2\pi$ (MHz)	$P_{\text{1ph}}$ (MHz)	$n_{\text{crit}}$	$t_k$ (ns)
1	5.092	39.2	0.5	5.1	7.062	5.35	0.16	102.9	36.1	14.65	186.9
2	4.404	6.4	0.5	2.9	7.102	5.63	0.07	81.7	14.1	19.07	177.6
3	5.000	21.3	0.2	3.8	7.152	6.62	0.12	103.6	35.9	12.93	151.1
4	4.560	11.8	0.4	3.3	7.197	7.43	0.06	89.9	21.7	15.47	134.6
5	5.165	21.0	5.7	31.9	7.254	12.00	0.07	105	63.9	6.93	83.3

Table 3.1: System parameters used in our simulations. Parameter values for the 5-qubit qubit chip based off a lab experiment focused on qubit-state discrimination.



# Chapter 4

## Machine Learning Background

### 4.1 Machine Learning

Machine learning is a field of algorithms that aim to automatically learn patterns from data. Building a model from a set of training data, machine learning algorithms can learn generalizable parameters that describe that data and then apply those parameters to a hidden test dataset. Algorithms that fall under the umbrella of machine learning range from simple linear regression to 175 billion parameter deep natural language processors [29]. There are three main families of machine learning problems.

#### **Supervised Learning**

In supervised learning, we train models using a fully labeled dataset to perform prediction on a set of unlabeled test data. This often means that supervised learning models require large amounts of diverse training data that is trained on repetitively. Common examples include regression, image classification, and time series prediction.

#### **Unsupervised Learning**

In unsupervised learning, the goal of the models is to find hidden structures of unlabelled data, often in terms of relating individual data points to each other. Common

use cases include clustering, auto-encoding, and anomaly detection. Because unsupervised learning environments do not have data labels, researchers often have to add extra steps to interpret results, such as assigning classes to learned clusters.

## **Reinforcement Learning**

The final family of machine learning algorithms is reinforcement learning. In reinforcement learning, we aim to have an agent learn how to map situations to actions to maximize an external reward signal. Importantly, this agent is forced to interact with an environment to explore new state-action pairs and potential rewards. The agent interacting with an environment is an important difference compared to supervised and unsupervised learning. Here we do not have a large pre-labeled training dataset, instead, new training inputs are directly observed from the environment.

The focus of this chapter and the rest of the thesis will be on reinforcement learning algorithms and their building blocks. Specifically, we will look at reinforcement learning algorithms backed by deep neural networks, giving us greater ability to optimize policies in large complex environments.

## **4.2 Reinforcement Learning**

Reinforcement learning is a subset of machine learning where an agent aims to maximize a reward by choosing the appropriate action at a given state. Rather than comparing predictions with labeled datasets like in supervised learning, reinforcement learning algorithms “learn” through the process of their agents directly interacting with the environment. Additionally, reinforcement algorithms do not need to know how the underlying system functions. Rather, they simply need to be able to approximate the reward for a state-action pair. The growth of reinforcement learning research and applications have been shown through success in complex games such as Chess and Go [8, 9], as well as robot automation and planning [10].

To best describe a reinforcement learning problem, we need to define four main components:

- The environment defines how our agent can operate. Specifically, the possible states an agent can be in, the possible actions an agent can take from a state, the transition model from one state to another, and the termination conditions. We often work in contexts where an agent has a limited view of the environment, often just the state they are in and the available actions from that state.
- The policy, which is what is directly or indirectly “learned”, governs how agents select actions in an environment given a current state. The policy may be as simple as a lookup table or predefined action logic. More often in complex environments, an optimal policy is optimized through approximation models like neural networks.
- A reward signal defines the agent’s goal in a reinforcement learning problem. Upon arriving in a new state, the environment provides a reward for taking the previous action, and thus the goal of the agent is to maximize the total reward while acting in that environment. Goals can vary in magnitude and frequency. In some problems like learning how to play Go, the reward signal is sparse, simply being 1 at the end of an episode if you win or 0 if you lose [8]. Conversely, some environments might provide frequent rewards to enable multiple policy considerations, such as having a small negative reward at each step to train an agent to take as few steps as possible to reach a goal.
- While the reward signal gives an agent the immediate reward for arriving at a state, it is also important for an agent to have an idea of the future value of arriving at a state in an environment. We encode this information in what is known as a value function, where the value is the amount of total reward an agent can expect given its current state and learned policy. A good value function is especially important in environments that might have many local maxima in terms of rewards, as an agent can better plan to find global maxima. Learning a good value function is obviously much harder than taking in an immediate reward, and highlights the necessity for an agent to thoroughly explore environments that are sufficiently large and/or complex.

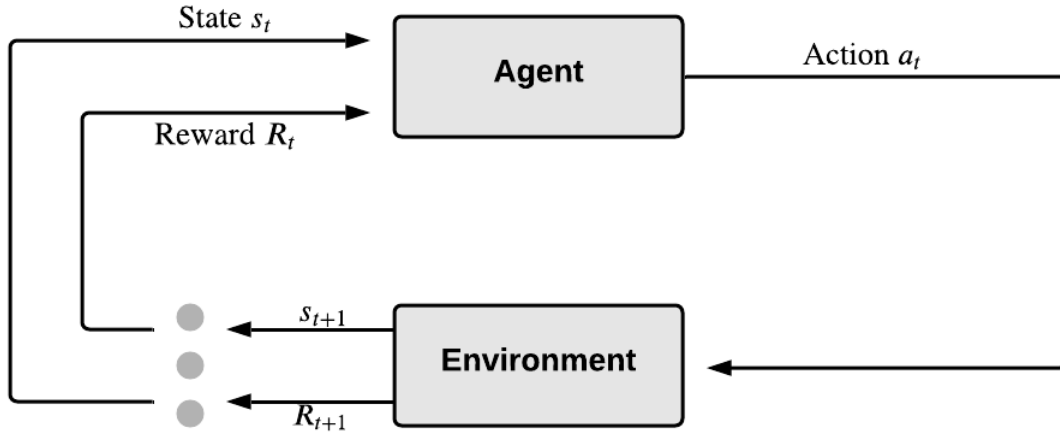


Figure 4-1: General flow of a Markov decision processes. Given a current state ( $s$ ) and reward ( $R$ ) at time step  $t$ , an agent chooses an action that is fed into the environment which produces the next state and reward at for time step  $t + 1$ , continuing the cycle until reaching a terminal state.

### 4.2.1 Markov Decision Processes

Reinforcement learning can be mathematically formalized in the form of finite Markov decision processes (MDP). At each time  $t$ , the black-box environment gives an agent its current state  $s_t$  and reward for arriving at that state  $R_t$ . Given that information and the information, it's learned about the environment, the agent then selects an action  $a_t$  that is sent to the environment to be performed. The cycle then continues with the agent returning the next state and reward  $s_{t+1}$  and  $R_{t+1}$  and continues until the agent reaches a terminal state in the environment. In a general MDP formulation, every transition is probabilistic. If we select an action  $a$  from state  $s$ , the next state  $s'$  and reward  $R$  come from a probability distribution  $P$ . Additionally, any step  $t$  is only dependent on the previous step  $t - 1$ .

### 4.2.2 Bellman Equation

As discussed, a key component in any agent's success is its ability to encode the value of particular states and actions in the environment. In the general MDP framework, the value function is often encoded in what is known as the Bellman equation [30].

The Bellman equation is defined as

$$V(s) = \max_a (R(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s')), \quad (4.1)$$

where the value of state  $s$  is determined by the available action  $a$  that maximizes both immediate reward  $R$  and future value discounted by factor  $\gamma$ . From this equation, we can find the optimal value function with value iteration and infer an optimal policy (Section 2.3) or directly optimize an optimal policy using policy iteration (Section 2.4).

### 4.2.3 Value Iteration

The first method we will use to find optimal policies in the MDP problems is value iteration. Here, we aim to learn the best value function  $V(s)$ , which can then be used to find a deterministic policy  $\pi'$  using the following algorithm

---

**Algorithm 1** Value Iteration [7]

---

Randomly initialize  $V(s)$  for all  $s \in S$ .  
Initialize  $\Delta > 0$  as a small threshold to determine if the value function has converged.  
**loop**  
     $\delta \leftarrow 0$   
    **for**  $s \in S$  **do**  
         $v \leftarrow V(s)$   
         $V(s) \leftarrow \max_a (R(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s'))$   
         $\delta \leftarrow \max(\delta, |v - V(s)|)$   
    **until**  $\Delta > \delta$   
 $\pi'(s) = \operatorname{argmax}_a (R(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s'))$

---

Here, an agent will get better at optimizing the value function as it gains more experience. This usually means that we need to visit many states multiple times to fully explore our environment. One key benefit though is that knowing an optimal value function allows one to infer an optimal policy simply by tapping into that value function.

#### 4.2.4 Policy Iteration

An alternative to value iteration is to directly optimize for an optimal policy using what is known as policy iteration, with the general algorithm below.

---

**Algorithm 2** Policy Iteration [7]

---

Randomly initialize  $V(s)$  and  $\pi(s) \in A$  for all  $s \in S$ .  
Initialize  $\Delta > 0$  as a small threshold to determine value function has convergence.

(a) Policy Evaluation

**loop**

$\delta \leftarrow 0$

**for**  $s \in S$  **do**

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a (R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s))V(s'))$

$\delta \leftarrow \max(\delta, |v - V(s)|)$

until  $\Delta > \delta$

(b) Policy Improvement

$policy-stable \leftarrow \text{true}$

**for**  $s \in S$  **do**

$old-action \leftarrow \pi(s)$

$\pi(s) \leftarrow \operatorname{argmax}_a (R(s, a) + \gamma \sum_{s'} P(s'|s, a)V(s'))$

**if**  $old-action \neq \pi(s)$  **then**

$policy-stable \leftarrow \text{false}$

**if**  $policy-stable = \text{false}$  **then**

    Go to (a)

---

The differences between policy iteration and value iteration are subtle but important. In policy iteration, we select actions from a current policy compared to computing the value of every available action and post-selecting an action. We are also always optimizing an optimal policy in policy optimization, instead of only finding an optimal policy a single time in the end once we've found an optimal value function. Policy iteration approaches are especially helpful in environments that have continuous action spaces, where it would be infeasible to compute a value for every state-action pair. However, value iteration approaches can be advantageous in discrete action space environments as it can be quicker to optimize [7]. We use one of each approach in our DRL algorithms found in Chapter 5 Section 3.

## 4.3 Deep Learning and Neural Networks

The driving force behind many of the advances in machine learning over the past decade comes from neural networks and what is known as “deep learning”. A neural network is a biologically-inspired network graph consisting of layers of connected nodes. The value of each node is the sum of a bias term and the product of each connected node from a previous layer and edge weight that connects the two nodes. That node value is then fed through a non-linear function and becomes an input for the next layer of nodes. The values of the bias term and edge weights are trainable parameters that are updated based on the performance goal of the problem at hand.

The ideas behind neural networks have been around since the 1940s, with research focus waning on and off during the 20th-century [31]. The explosion in research that has fueled so many advancements in fields such as computer vision and robotic control has come from what we know as deep learning. With computational and memory advancements, we can now store and train networks that have many hidden layers (“depth”). By increasing depth, we increase the number of learnable non-linear transformations and thus allow these networks to outperform many existing systems. A helpful comparison is looking at an  $n$ -dimensional linear binary classifier, which can be viewed as a neural network that only has an  $n$  node input layer and single node output layer. If our dataset is not linearly separable in the beginning, we might look to perform some manual data transformation before optimizing a line of separation. With neural networks, we can use any number of hidden layers to allow our system to learn what the ideal non-linear data transformations are, instead of relying on human subject experts and trial and error.

### 4.3.1 Feed-Forward Neural Networks

In this thesis, we focus on the most basic neural network architecture called a feed-forward neural network (FNN), or multi-layered perceptron (MLP). As we see in Fig 4-2(a), the FNN is comprised of layers of connected neurons that map in an input vector to an output value/vector. As shown in Fig. 4-2(b), the value of any node  $i$  in

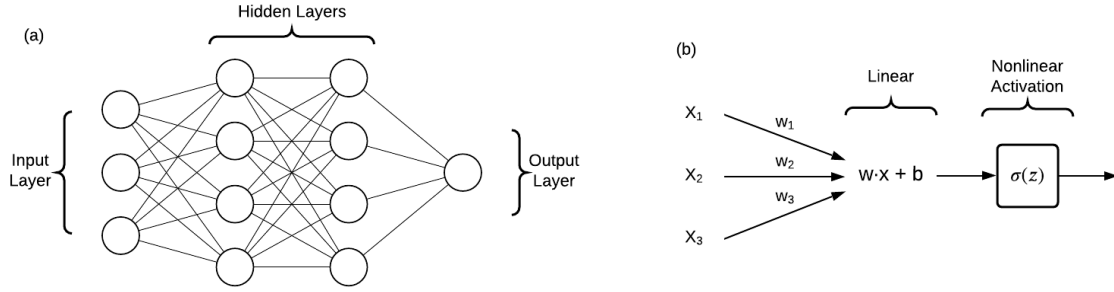


Figure 4-2: (a) Basic feed-forward neural network architecture with an input layer of 3 nodes, 2 hidden layers with a width of 4 nodes each and single output layer node. (b) The value of each node in a neural network not in the input layer is determined by finding the sum of a bias term and the dot product between the previous layer's node values and the inter-layer weight values. This computed value is then fed into a non-linear activation function.

layer  $l \in L$  in our neural network can be defined as

$$a_i^l = \sigma \left( \sum_j w_{(i,j)}^l a_j^{l-1} + b_i^l \right) = \sigma(z_i^l), \quad (4.2)$$

where  $w_{(i,j)}^l$  is the weight between nodes  $i$  and  $j$  between layers  $l - 1$  and  $l$ ,  $b_i^l$  is the bias weight, and  $\sigma$  is an activation function (see 4-3). These activation functions are critical to neural networks because they introduce non-linearities into our neural networks, providing a relatively efficient means for the networks to identify complex data patterns. Here we describe the most common activation functions used in FNNs.

## Linear

The linear activation function (Fig. 4-3) simply returns the node value as it was passed into the activation function. We almost always do not use linear activation functions in the hidden layers as they do not introduce non-linearity to the network. However, they are commonly found in output layers that require continuous values such as in value prediction contexts and regression contexts. The linear activation function and its derivative are given by

$$\sigma(z) = z \quad (4.3)$$

$$\sigma'(z) = 1 \quad (4.4)$$

## ReLU

The rectified linear unit activation function (ReLU) (Fig. 4-3) is typically the default activation function used in the hidden layers of FNNs. The ReLU function is favored heavily because while it introduced non-linearity, it is still very similar to that of a linear transformation, and thus can benefit from being a good generalizer and from easy to optimize with gradient-based methods. The ReLU activation function and its derivative are given by

$$\sigma(z) = \max(0, z) \quad (4.5)$$

$$\sigma'(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases} \quad (4.6)$$

## Sigmoid

Sigmoid activation functions (Fig. 4-3) are commonly used in the output layer of neural networks when we are performing binary classifications. Typically, we find that the sigmoid output layer is a single node, however, some problems do have multiple output nodes when performing multiple binary classifications in a single forward pass. The sigmoid activation function and its derivative are given by

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (4.7)$$

$$\sigma'(z) = \left( \frac{1}{1 + e^{-z}} \right) \left( 1 - \frac{1}{1 + e^{-z}} \right) \quad (4.8)$$

## Softmax

The softmax activation function is similar to the sigmoid activation in that it maps the output node value to a probability, but now working in a setting where our output layer has multiple nodes. Softmax is widely used for multi-class prediction and reinforcement learning applications where we can map output nodes to available

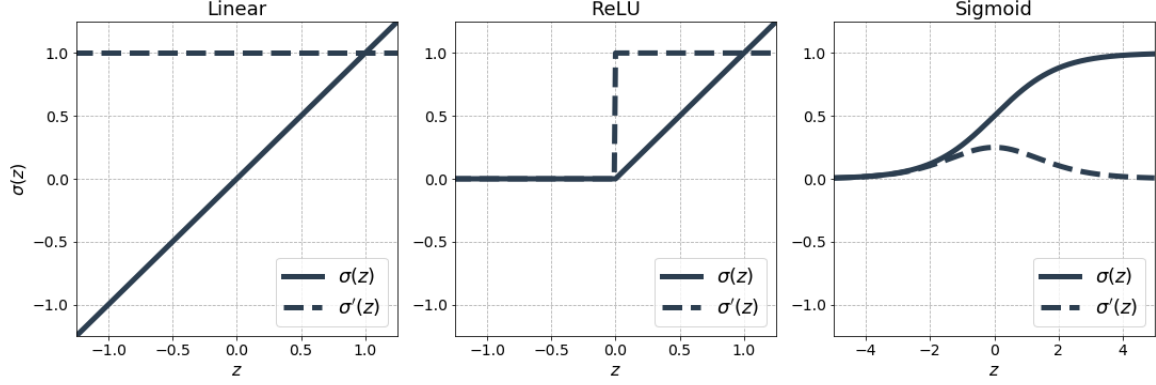


Figure 4-3: Common activation functions and their derivatives found in FNNs, including the linear, ReLU and sigmoid activation functions.

actions. The softmax activation function and its derivative are given by

$$\sigma(z) = \frac{e^z}{\sum_i e^i} \quad (4.9)$$

$$\sigma'(z) = \left( \frac{e^z}{\sum_i e^i} \right) \left( 1 - \frac{e^z}{\sum_i e^i} \right) \quad (4.10)$$

### 4.3.2 Cost Functions and Regularization

The most essential component of any machine learning system is how we train it. The first step in training a deep neural network is to define a loss function. If we follow the architecture in Fig. 4-2(a) and state the single output node has a linear activation function, we can consider the network to be making a continuous prediction. In that case, a common cost function is the mean squared error over  $n$  examples,

$$C(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i(\theta))^2. \quad (4.11)$$

Like any optimization technique, neural networks are susceptible to overfitting, especially because neural networks tend to have many more trainable parameters. There are several potential regularization techniques we can use to prevent overfitting.

One example is to directly include a regularizer in the cost function as

$$C(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i(\theta))^2 + \Omega(\theta), \quad (4.12)$$

where  $\Omega(\theta)$  (L2 regularizer) is the squared sum of all model weights. Applying this regularization terms penalizes the model for updating to large individual weight values, which can be a major cause of overfitting. Another regularization technique we use is batch normalization [32]. Here, instead of feeding our neural network one input at a time, we create a minibatch of  $n$  inputs and normalize the input values in respect to that batch.

### 4.3.3 Training Neural Networks

Now that we have defined a neural network architecture and cost function, we can move to actually train the network. We update the network weights using a gradient descent-like algorithm that computes the cost functions derivative concerning all trainable weights in the network. Since deep neural network weights can easily number in the tens of millions in some applications, it's impractical to compute every single gradient individually. Instead, we use a method called backpropagation ("backprop") that allows us to take advantage of the network's layer structure and generate these gradients by essentially just recursively applying the chain rule for partial differentiation [33].

A particular weight or bias in a neural network is updated such that

$$w := w - \eta \frac{\partial C}{\partial w}, \quad (4.13)$$

$$b := b - \eta \frac{\partial C}{\partial b}, \quad (4.14)$$

where  $C$  is our cost function and  $\eta$  is the learning rate for the training optimization loop. To calculate  $\frac{\partial C}{\partial w}$  and  $\frac{\partial C}{\partial b}$  for a particular layer, we rely on using the chain rule and four key equations.

1. For our first key equation, we use our cost function to define the error of a node  $i$  in any layer  $l$  as the change in the cost function concerning the node value  $z_i^l$  such that

$$\Delta_i^l = \frac{\partial C}{\partial z_i^l} = \frac{\partial C}{\partial a_i^l} \sigma'(z_i^l). \quad (4.15)$$

2. Similar to the first equation, we can also define the error of a node  $i$  in any layer  $l$  as the change in the cost function concerning the bias term  $b_i^l$ . Here we can set  $\partial b_i^l / \partial z_i^l = 1$  and

$$\Delta_i^l = \frac{\partial C}{\partial z_i^l} = \frac{\partial C}{\partial b_i^l} \frac{\partial b_i^l}{\partial z_i^l} = \frac{\partial C}{\partial b_i^l}. \quad (4.16)$$

3. Working recursively, any error from a node in layer  $l$  only depends on node errors in layer  $l + 1$ . Thus we can use the chain rule to find

$$\Delta_i^l = \frac{\partial C}{\partial z_i^l} = \sum_j \frac{\partial C}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_i^l} = \sum_j \Delta_j^{l+1} w_{(i,j)}^{l+1} \sigma'(z_i^l). \quad (4.17)$$

4. Finally, we can differentiate the cost functions concerning the weights of the network between layers  $i$  and  $j$  with

$$\frac{\partial C}{\partial w_{(i,j)}^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{(i,j)}^l} = \Delta_i^l a_j^{l-1}. \quad (4.18)$$

Using these four equations, we start by passing through an input into our network and calculating all  $a_i^l$  and  $z_i^l$  values. Next, we can directly calculate the output layer  $L$  error using Eq. 4.15. With this value, we can move back across the network and calculate all recursively calculate all subsequent errors. Finally, the values of all errors  $\Delta_i^l$  are used to find all  $\frac{\partial C}{\partial w}$  and  $\frac{\partial C}{\partial b}$  and update our network parameters.

## 4.4 Deep Reinforcement Learning

Now that we have an understanding of reinforcement learning and deep neural networks, we look to combine the two ideas with deep reinforcement. As we see in reinforcement algorithms like value iteration (Algo. 1) and policy iteration (Algo. 2),

finding ideal agent policies often requires an extremely high number of episodes of training in an environment to converge on a policy. With large and complex environments, it is impractical to try to approach these problems iteratively. There are too many possible states and actions to store value functions in memory and there are too many possible states to explore to ensure an optimal policy. To remedy this issue, as pulse shaping is an extremely large and complex environment, we look to utilize deep neural networks as part of our reinforcement learning algorithms. Typically, this means using those neural networks as approximators for a value function or policy distribution. Instead of storing every state's value function result in memory, we can use a neural network to approximate the future value of that state and then train the neural network on the real value that is seen later. Similarly on the policy side, instead of constantly updating a policy model, we can use a neural network to approximate the distribution of models that are trained based on the generated policy's performance. We will explore two of these deep reinforcement algorithms, Deep Q Networks, and Proximal Policy Optimization in Chapter 5.



# Chapter 5

## Methodology

### 5.1 Conventional Approach

As we’ve described earlier in Chapter 3, the conventional approach to dispersive readout pulse shaping is to use a sufficiently long rectangular pulse. Because the resonator’s frequency is shifted by the qubit state, that rectangular pulse can probe that frequency shift and thus allow us to discriminate the underlying qubit states. However, this process is slow. First, a rectangular pulse sent to the resonator frequency often needs to be fairly long (multiple microseconds) to populate and stabilize the resonator. Second, once we’ve finished probing, it’s important to get the resonators back to the vacuum state. With the rectangular pulse method, we simply stop applying the pulse and passively allow the photons to naturally decay out of the resonator.

Here we look at two approaches to generate readout pulse shapes. First, we reimplement the existing Cavity Level Excitation and Reset (CLEAR) pulse [34] and extend it to be optimizable in a multiplexed readout environment. Second, we use two deep reinforcement learning techniques to generate readout pulse shapes with much finer amount of control compared to other available methods.

## 5.2 CLEAR Pulse

### 5.2.1 Introduction

In circuit QED devices, a large research focus has surrounded improving readout speed and fidelity (qubit-state classification accuracy in this case), including the development and improvement of Purcell filters and quantum-limited amplifiers [35, 36, 37]. From a pulse shaping optimization perspective, the most successful approach has been found through the Cavity Level Excitation and Reset (CLEAR) pulse [34]. The CLEAR pulse relies on a set of two short segments attached to the front and end of a typical rectangular pulse that helps populate and depopulate the resonator with photons quicker than conventional methods (Fig. 5-1). For a single qubit, the two amplitude values of the photon injection and reset components can be empirically optimized depending on the target photon population number  $\bar{n}$  of its subsequent drive power  $P_{\text{norm}} = P/P_{\text{1ph}}$  where  $P_{\text{1ph}}$  is the drive power required for a steady-state photon population of  $\bar{n} = 1$ . The pulse amplitudes corresponding to different photon target values need to be optimized separately because higher drive powers can force the cavity outside of the linear regime where the steady-state photon number  $n_0$  is now dependent on the prepared qubit state.

### 5.2.2 Scaling to Multi-Qubit Systems

One of the key benefits of the CLEAR pulse for single-qubit readout is the ease of optimization, as only two amplitude segments need to be tuned at a given time, which can be achieved through a simple empirical optimization. When scaling to  $n$ -qubit multiplexed readout, this optimization becomes a  $2n$ -parameter optimization for both the injection and reset portions of the pulse, thus requiring a more complex optimization protocol that can handle the higher parameter number and more complex optimization space due to the various forms of cross-talk introduced in multiplexed readout (Chapter 3 Section 3).

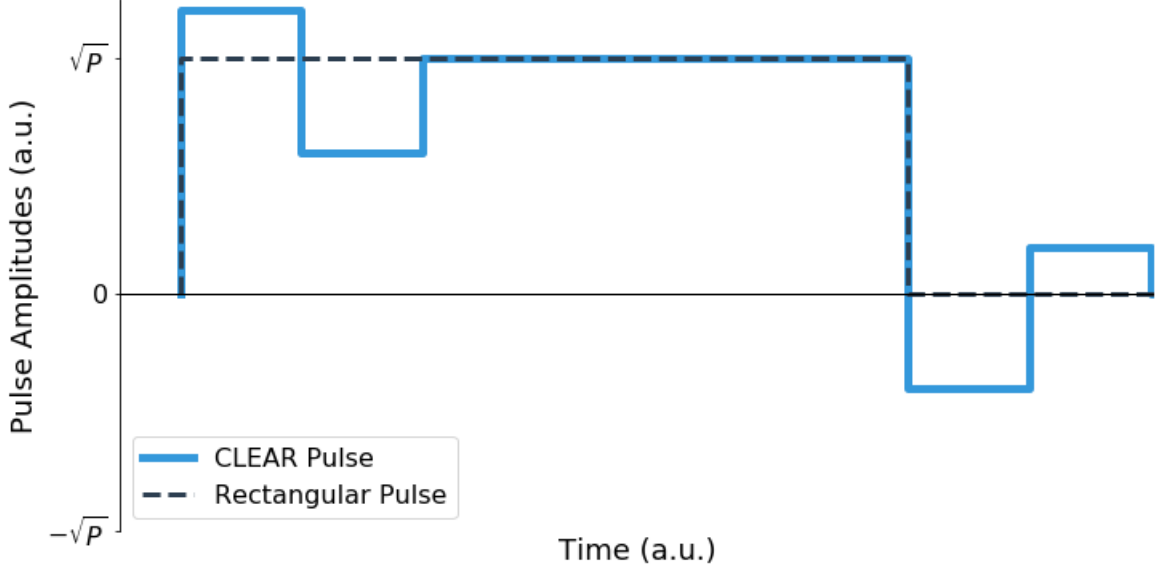


Figure 5-1: Arbitrary CLEAR and rectangular pulse shapes. Two amplitude segments at the beginning of the pulse help inject photons into the resonator quickly, allowing the resonator to reach a steady photon count in a shorter time. Two amplitude segments are also added to the end of the pulse to actively reset the resonator to the vacuum state.

### 5.2.3 Bayesian Optimization

We handle this increase in optimization complexity with a technique called Bayesian optimization. Bayesian optimization is an effective black-box optimization scheme that attempts to build a probabilistic model of an underlying function that makes it simple to both identify good parameters to sample from (explore) and good parameter candidates to maximize reward (exploit). Once we have an optimized model for our underlying black-box environment, selecting an optimal parameter set is simply

$$x^* = \operatorname{argmax}_{x \in \mathcal{X}} f(x). \quad (5.1)$$

To build this underlying model, we use Gaussian Processes (GPs), which are a probability distribution over a collection of possible functions fit to a set of points. A Gaussian process can be defined as a normal distribution with a mean vector  $\mu$  and a covariance matrix  $\Sigma$ , where the expected value of the underlying function is also equal to  $\mu$ . Additionally, the variance of each test point  $i$  can be found along

the diagonal of the covariance matrix  $\Sigma$ . As we see, increasing the number of initial and optimization points increases the dimension of  $\mu$  and  $\Sigma$ , often resulting in the optimization algorithm progressively getting slower over time (Fig. 5-2).

A key component to Gaussian processes is defining a kernel that builds our covariance matrix  $\Sigma$ . In this work, we use the common Radial Basis Function (RBF) kernel where

$$K(x, x') = \exp\left\{\frac{\|x - x'\|^2}{2\sigma^2}\right\}, \quad (5.2)$$

with  $\sigma$  being a tunable parameter that increases or decreases the variance.

For our optimizer’s acquisition function, we use a technique called GP Upper Confidence Bound (GP-UCB) to compute the ideal next sample [38]. GP-UCB exploits the upper confidence boundary on the underlying Gaussian posterior such that we can simply choose the maximum  $x$  in

$$\alpha_{\text{UCB}}(x) = \mu(x) - \kappa\sigma(x). \quad (5.3)$$

Here,  $\kappa$  is a tunable parameter that adjusts how much our acquisition function favors exploring potential high reward points. With complex underlying optimization spaces, it’s often advantageous to have higher  $\kappa$  to be able to explore as much of the environment as possible.

---

**Algorithm 3** Bayesian Optimization

---

Place a Gaussian process prior on  $f$   
 Observe  $f$  at  $x_0$  initial uniform random points  
 Update posterior probability distribution on  $f$  based on initial points  
**for**  $i = 1, 2, \dots$  in  $N$  **do**  
     Select  $x_i$  as the maximizer of the chosen acquisition function.  
     Calculate  $y_i = f(x_i)$   
     Update posterior probability distribution on  $f$  based on *all* points.  
 Return largest observed  $f(x)$

---

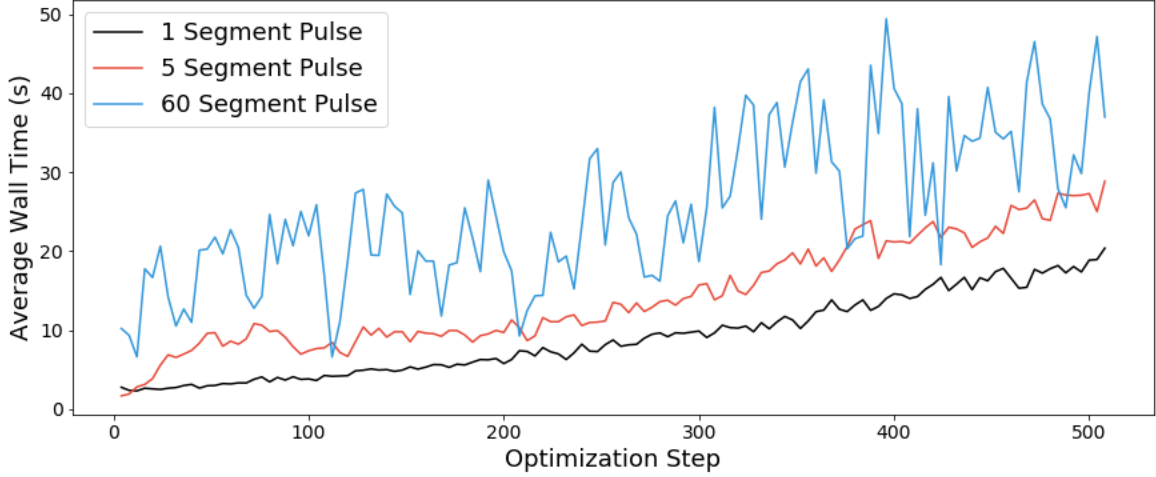


Figure 5-2: Wall-clock time needed per optimization step when optimizing different numbers of pulse segments. Regardless of the number of segments being optimized, the length of each optimization step generally increases as the optimization step number increases as we expect from the Bayesian optimization algorithm. Additionally, increasing the number of amplitude segments needing to be optimized also results in requiring more optimization time per step throughout the entire optimization process.

## Scaling

One of the disadvantages of Bayesian Optimization, and why it is not used to generate pulses that contain more than two segments, is its poor scaling. Specifically, every update to the posterior probability distribution requires using every parameter-observation pair, including the initial random points. More complex environments will likely require more optimization steps to find a suitable set of parameters. This problem is exaggerated when working in a multi-qubit system where each additional amplitude segment creates 5 additional parameters needing to be optimized, and thus creates an even more complex optimization environment. The effects of increasing the number of parameters in a Bayesian Optimizer can be seen in Fig. 5-2.

## 5.3 Deep Reinforcement Learning Generated Pulses

### 5.3.1 Deep Q-Networks

#### Q-Learning

The first deep reinforcement learning algorithm we implement and experiment with is the Deep Q-Network (DQN) algorithm [39]. DQN's are an extension of the popular Q-Learning algorithm that allows us to explore larger state and action spaces, as well as take advantage of modern deep neural network techniques and hardware optimizations [40].

In tabular Q-Learning settings, we aim to find an optimal policy by creating a Q-table that holds a value for each state-action pair. Each table entry signifies the current value of a pair by acting in that given state, as well as the agent's future value by making decisions from that new state-action pair. More specifically, the Q-table is updated with a variation of the Bellman equation (Eq. 5.4), where the state-action pair Q-value of  $Q_{\text{new}}(s_t, a_t)$  is updated by summing the old value with optimal next step in the table. Using value iteration and enough training iterations, optimal values for the Q-table can be eventually computed. When the table is complete, an agent can simply find an optimal solution by performing table look-ups and selecting the action that produces the largest Q-value at each state.

$$Q_{\text{new}}(s_t, a_t) = Q(s_t, a_t) + \alpha * (\text{Reward}(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a)) \quad (5.4)$$

#### Exploration vs. Exploitation

One of the key challenges in any reinforcement learning is striking the correct balance between exploration and exploitation. Particularly, when we have large and complex environments, we must be able to explore different parts of that environment so we do not end up simply optimizing to a local maximum based on a random starting point, a problem that other optimization schemes in quantum control like GRAPE encounter [41]. In DQN, we actually handle this with a simple approach called  $\epsilon$ -

greedy action selection. Here, we choose a random action with probability  $\epsilon$  or an action generated by our training policy with probability  $1 - \epsilon$ . A common strategy is to start with a high  $\epsilon$  value at the beginning of the training that eventually descends to a value very close to zero, allowing the algorithm to “explore” in the beginning and gradually “exploit” as training continues.

## Deep Q-Networks

A major issue with tabular Q-Learning approaches is the reliance on storing the Q-table in memory, as well as the algorithm’s necessity to explore many state-action pairs. This becomes infeasible for environments with large and precise state and action spaces, such as the case with multi-qubit pulse shaping. Deep Q-Networks replace the Q-table with a deep neural network that takes in a state input and outputs the approximated Q-values for each action (Fig. 5-3). To train the network, we store past network predictions in a replay buffer that is periodically sampled uniformly or by prioritizing samples with higher rewards [42]. The underlying neural network is trained by minimizing the loss function,

$$L_i(\theta_i) = \hat{E}_{(s_i, a_i, r_i, s_{i+1}) \sim \mathcal{D}}[(y_i - Q(s_i, a_i; \theta_i))^2] \quad (5.5)$$

For pulse shaping, we define our state as being the partially constructed pulse shape and our action space being the discrete set of all available amplitudes. Because DQN’s select one action at a time, the pulse shape is built sequentially, where  $R = 0$  automatically for any pulse incomplete pulse shape. While this creates to a sparse reward signal, the eventual value of good pulse shapes can be adequately encoded by using a high discount rate ( $\gamma > 0.99$ ). This algorithm requires us to have a discrete action space as the network output predicts a Q-value for each possible action. Discretization can be helpful in that the physical implementation using an AWG also has a discrete set of amplitudes, and thus the model can mirror that action space. Unfortunately, a large action set size also makes the neural network approximator more difficult and time-consuming to train due to a high number of output nodes.

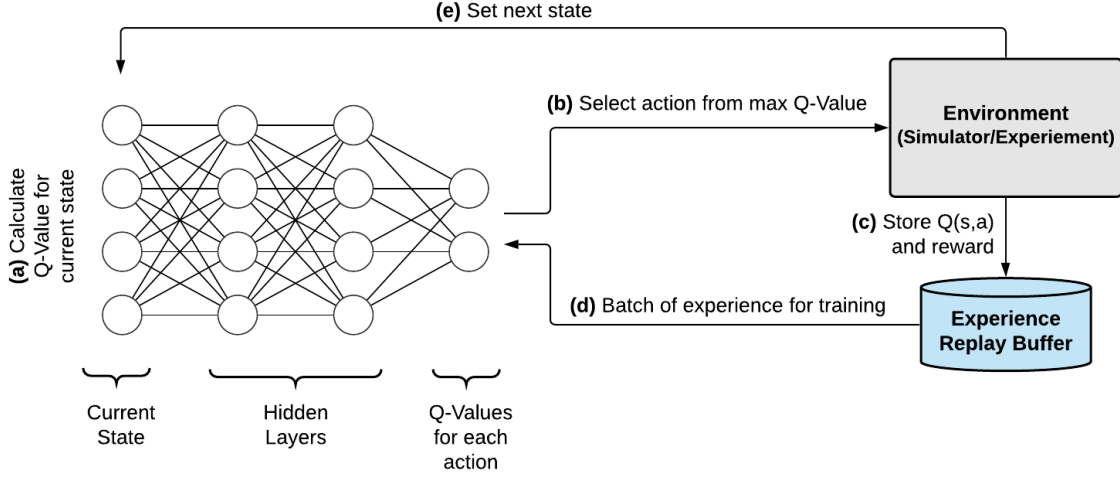


Figure 5-3: A simple Deep Q-Network architecture where we are trying to construct a single 4 segment pulse with 2 possible amplitude options. (a) A state is provided as the input for a deep fully-connected neural network. The neural network’s outputs approximate the Q-value for each possible action according to the state input. (b) The action with the maximum associated Q-value is selected, appended to pulse, and sent to interact with the simulated or physical environment. (c) The state-action pair’s actual Q-value is calculated in this environment and is saved in an experience replay buffer. (d) A batch of expected and actual Q-values with their state-action pairs are used to train the neural network. (e) The new state generated by the environment is passed back to the input of the neural network, restarting the process.

---

**Algorithm 4** DQN’s with Experience Replay [39]

---

Create an experience replay buffer  $\mathcal{D}$  of size  $N$   
Initialize Q-value function neural network with random weights  
**for**  $episode = 1 \dots M$  **do**  
    **for** time step  $t = 1 \dots T$  **do**  
        Select random action with probability  $\epsilon$ ,  
        or select action  $a_t = \operatorname{argmax}_a Q(s_t, a; \theta)$ .  
        Perform action  $a_t$  in environment and observe reward  $r_t$  and next state  $s_{t+1}$   
        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $\mathcal{D}$ .  
        Uniformly samples batch of transitions from  $\mathcal{D}$ .  
        For each sample  $j$  in batch of transitions, set  $y_j = r_j + \gamma \max_a Q(s_{j+1}, a; \theta)$ ,  
        where all future  $Q$  values are equal to 0 at terminal states.  
        Perform gradient descent to minimize loss and update weights.

---

## Scaling

For scaling to multi-qubit pulse engineering, we looked at a few for modifying the DQN approach. The key underlying challenge is that for any time step  $t$ , the algorithm now needs to select  $n$  pulse amplitudes where  $n$  is the number of qubits in the system. One approach that has been used for some video game environments is to create every combination of paired actions to become the new action space [39]. In pulse shaping that would mean the action space size would now be  $a^n$  where  $a$  is the number of discrete amplitude values available to be chosen. Even if we restrict that set to  $2^{10}$  amplitude values, that still results in an output layer with  $2^{50}$  nodes, which is too large for any system both in terms of training that large of a network and even storing it in memory. Another approach we looked at was to have separate networks for each pulse. By sharing other pulses state information in each other's input layer, we could train the networks simultaneously and but keep the size of each network similar to that of a single pulse environment. Unfortunately, this approach was unable to converge in its training in a reasonable time-frame that could even be practical for real-world applications.

### 5.3.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a state of the art deep reinforcement learning algorithm designed to directly optimize policy objectives [43]. Here, we use deep neural networks to learn an optimal distribution of policies for a given objective. The policy distribution is sampled several times and the resulting data is used to generate a better distribution of policies for the next step. PPO can generate entire multiplexed pulse shapes in a single forward pass through the policy network, as each output node is now a continuous variable that maps to a particular pulse segment. This is a huge advantage compared to DQN where we had to cycle through the network  $n$  times for an  $n$ -segment long pulse. Additionally, because we work in the continuous output regime, we can allow for much finer control of the pulse amplitudes without blowing up our neural network size as we saw with DQNs.

## Policy Gradients

PPO belongs to a class of reinforcement learning algorithms called policy gradient methods, where the goal of the algorithm is to directly learn optimal policies, rather than infer them from optimal value functions like in Deep Q-Networks. The use of a value function may still be useful, as we will see in the PPO loss function, but it is not required for policy gradient algorithms. The loss of a general policy gradient algorithm is defined as

$$L^{\text{PG}}(s, a, \theta) = \hat{E}[\log \pi_{\theta}(a|s)\hat{A}], \quad (5.6)$$

where the advantage function  $\hat{A}$  is the absolute difference between the actual value and expected value that we approximate ahead of time. Additionally,  $\log \pi_{\theta}(a|s)$  are the log probabilities of taking each action. To minimize loss, we need to both be able to accurately predict the value of selecting action  $a$  and also be confident in that action prediction.

## Actor-Critic

PPO utilizes what is known as an actor-critic approach, in which an “Actor” agent uses the current state to approximate an optimal continuous action, while the “Critic” agent approximates the expected value for the new state-action pair. In this context, our value approximator network is used in the context of the advantage function  $\hat{A}$ , where being able to predict the correct value minimizes  $\hat{A}$ . Actor-critic methods compare well to single action-value methods like DQN because they require less computation to operate in a much larger action space and because the separation into two processes increases overall stability [7]. Similar to DQN, we will use deep neural network approximators for the Actor and Critic due to our large state and action set size.

## PPO

We can extend the policy gradient and actor-critic approaches to form the PPO algorithm. In PPO, instead of keeping a buffer of experience that we sample from to train our networks, we simply train on the most recent example and then discard that experience. This makes the algorithm much more sample efficient, but also susceptible to taking disadvantageously large step sizes that can force our policy distribution away from an optimal set of policies. The loss function of PPO becomes

$$L^{\text{PPO}}(\theta) = \hat{E}[\min(r(\theta)\hat{A}, \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A})], \quad (5.7)$$

$$r(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta, \text{old}}(a|s)}. \quad (5.8)$$

To remedy the issue of losing a good distribution of policies, there are two main alterations PPO includes compared to general policy gradients. First, we introduce the idea of “trust policy regions” with by using  $r(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta, \text{old}}(a|s)}$  [44]. Here, instead of sampling new pulses based on a blind training update step, we actually sample pulses from the previous policy distribution. This allows us to see which particular pulses are having more success at a more constrained scale and train our network more intelligently. To further restrain our network update size, we also introduce a clipping term, which allows us to train our network based on samples taken in a range that can be expanded/reduced by the tunable parameter  $\epsilon$ .

---

### Algorithm 5 PPO for Pulse Generating Training [43]

---

Place a Gaussian process prior on  $f$   
 Observe  $f$  at  $x_0$  initial uniform random points  
 Update posterior probability distribution on  $f$  based on initial points  
**for**  $i = 1, 2, \dots$  in  $N$  **do**  
   Use the existing policy  $\theta_{old}$  to generate  $N$  pulses.  
   Compute the advantage estimates  $\hat{A}_1, \dots, \hat{A}_N$  for the generated pulses.  
   Optimize  $L^{\text{clip}}$  with respect to  $\theta$ , using  $K$  epochs and a minibatch size  $M \ll N$   
   Update  $\theta_{old} \leftarrow \theta$

---

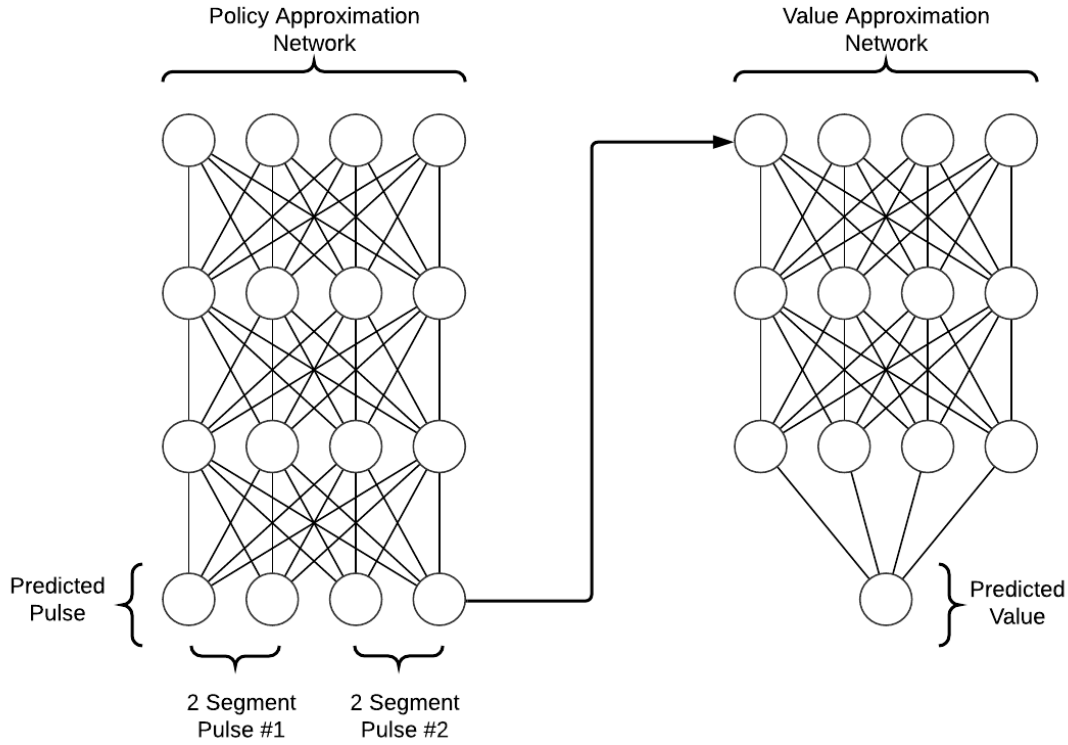


Figure 5-4: A simple Proximal Policy Optimization architecture where we are trying to build two 2-segment pulses where any bounded continuous amplitude can be chosen. We use two deep fully connected neural networks. The first is the policy network that aims to approximate an optimal policy (the best pulse shape). The second is the value network other that takes in the predicted pulse from the policy network and predicts the value of that pulse in the environment.

### 5.3.3 Other Approaches Considered

In addition to the DQN and PPO deep reinforcement learning approaches, we also briefly experimented with two other deep reinforcement learning algorithms in our simulated environments.

#### Deep Deterministic Policy Gradient

The first additional algorithm we looked at was the Deep Deterministic Policy Gradient (DDPG) algorithm, a model-free algorithm that can learn optimal policies in high dimensional and continuous action spaces [45]. A key difference between DDPG and PPO is that DDPG is an “off-policy” approach where similarly to DQN’s, it collects a

running history of pulses and their rewards, which is then sampled for training. PPO differs in that it doesn't store a running history. Instead, each generated pulse and the subsequent reward is used to train the model once and then is discarded, making PPO more sample efficient.

## **AlphaZero**

AlphaZero is a relatively new approach that has gained fame for outperforming master human players in games like Chess and Go without human game play samples for training [9]. While this project will not allow us to take advantage of self-play applications that powered its gameplay success, the method still provides a great tool for optimal control applications. In fact, AlphaZero was the critical algorithm behind the most consequential work in combining deep reinforcement learning and quantum optimal control [11] thus far.

AlphaZero employs a deep neural network in conjunction with deep look-ahead in a guided tree search, which allows for a predictive hidden-variable approximation of the state-action pairs. This search, a Monte Carlo Tree Search (MCTS), enables the algorithm to look many steps ahead in the environment, while DQN and DDPG are limited to a single-step look ahead. Additionally, the MCTS method allows for more efficient exploration of state-action pair space as it is guided probabilistically. DQN and DDPG on the other hand inefficiently rely on random action selection for the initial segments of training to explore the state-action pair space. While this MCTS approach does have these advantages, it also ended up requiring many more simulations per model update than that of any other approach. Because the training lacked efficiency, the training was overall too slow for practical implementation.



# Chapter 6

## Resonator Reset

### 6.1 Introduction

The first area of the multiplexed superconducting qubit readout we look at is resonator reset. More specifically, we ask how quickly can photons be removed from the readout resonators following a qubit state measurement. Fast readout resonator reset is critical as the conventional method of simply waiting for photons to decay from the resonator is time-consuming and often prohibitive to tasks like quantum error correction that requires quick operations following a measurement [5].

Typically, readout resonator reset is realized by simply waiting a fixed time long enough to ensure a majority of photons from all resonators have naturally decayed. The expected photon number in a resonator  $r$  over time is simply  $n_r = e^{-\kappa t} P_{\text{norm}}$  [34], where  $P_{\text{norm}}$  is the ratio between the applied power and the power required to inject 1 photon into a given resonator. In this chapter, we aim to optimize the end of a readout pulse to reduce the time required to remove photons from all readout resonators. We require the optimized reset pulse component to be unconditional on the underlying qubit states. We will compare the conventional method of letting photons naturally decay to that of the optimized CLEAR ring down pulse segments and to that of a deep reinforcement learning backed active reset pulse.

## 6.2 Optimization Setup

To optimize ideal pulse shapes, we must first define our optimization space and setup. Since we are only concerned with resonator reset in this chapter, we can define our black box reward signal as simply being the negative sum of photons remaining in the resonators. Similar to the work that first identified the CLEAR pulse, we set a target photon number of 0.10 as a lower boundary per resonator to account for any amount of coherent noise and thermal excitation that we may experience in experiments. The reward function for the rest of readout resonators  $R$  is defined below as:

$$\text{Reward} = \begin{cases} 0, & \text{if } \forall r \in R, n_r \leq 0.10 \\ -\sum_{r \in R} n_r, & \text{otherwise} \end{cases} \quad (6.1)$$

For populating the resonator(s) before reset, we apply a sufficiently long rectangular pulse at a drive power  $\sqrt{P}$  where  $P = P_{\text{norm}}P_{\text{1ph}}$ , in order to inject  $P_{\text{norm}}$  photons into the resonator(s). For all forthcoming simulated results, we set  $P_{\text{norm}} = 4$ . In our simulations, that rectangular pulse length is 3 microseconds to ensure stable photon counts in our 5 qubit setup. This is longer than we may really need in an experimental environment, but in simulation, it ensures the photon counts are fully stabilized.

For our optimized reset pulse shapes, a total of  $2^{10}$  amplitudes values between  $-\sqrt{P}$  and  $\sqrt{P}$  can be selected by the optimization methods. For methods that produce amplitudes in a continuous space like CLEAR and PPO, the outputted continuous amplitude is simply mapped to the closest available discrete amplitude. For all pulses used in the simulation, we convolve the generated pulse shapes with a Gaussian kernel ( $\sigma = 5$ ) to encompass the effects of a non-ideal step response [46, 47].

For CLEAR pulses generated using Bayesian optimization, we use 250 random initial points and up to 2,500 optimization points. For PPO generated pulses, we allow up to 51,200 training episodes, as each episode is much quicker compared to Bayesian optimization.

## 6.3 Single Qubit Simulation

### 6.3.1 Results

We start with a single qubit resonator reset setup, comparing natural photon decay following a rectangular pulse, a CLEAR pulse, and pulses generating using DQN and PPO. For these single qubit simulation results, we use the qubit and resonator parameters from Qubit 1 in Tab. 3.1 and ignore the effects of all other qubits and resonators. Here we find passive natural decay requires us to wait for 700 ns for the photon count to reach the target population count.

The optimized pulse shapes can be seen in Fig. 6-1(a). As expected, with the absence of other drives and qubit resonator pairs, the ideal pulse shape mirrors the CLEAR pulse. The first half of each pulse is largely at the most negative amplitude in the available range, while the second of the pulse is largely at the most positive available amplitude. This pattern agrees with previous work in reset and injection that has found that overshooting a particular target (reset or photon inject), followed by a correction is often faster than the typical rectangular pulse followed by turning off the pulse [34, 48, 49].

The DQN pulses and PPO pulses are 250 ns long, while the CLEAR pulse is 260 ns. This constitutes a 64.3% and 62.9% time decrease respectively, or around 2.41 and 2.35 photon decay times. The extra 10 ns is an artifact of the CLEAR pulse requiring equal length amplitude segments, and cutting that CLEAR pulse 10 ns short would still allow us to reach our reset target. We see the evidence of this in Fig. 6-1(b) where the time-dependent photon counts for the CLEAR pulse and the reinforcement learning-backed methods are practically identical. Additionally, these photon counts show how the initial negative amplitude segment decreases quickly but overshoots, but the second positive amplitude compensates and gets the photon count to zero.

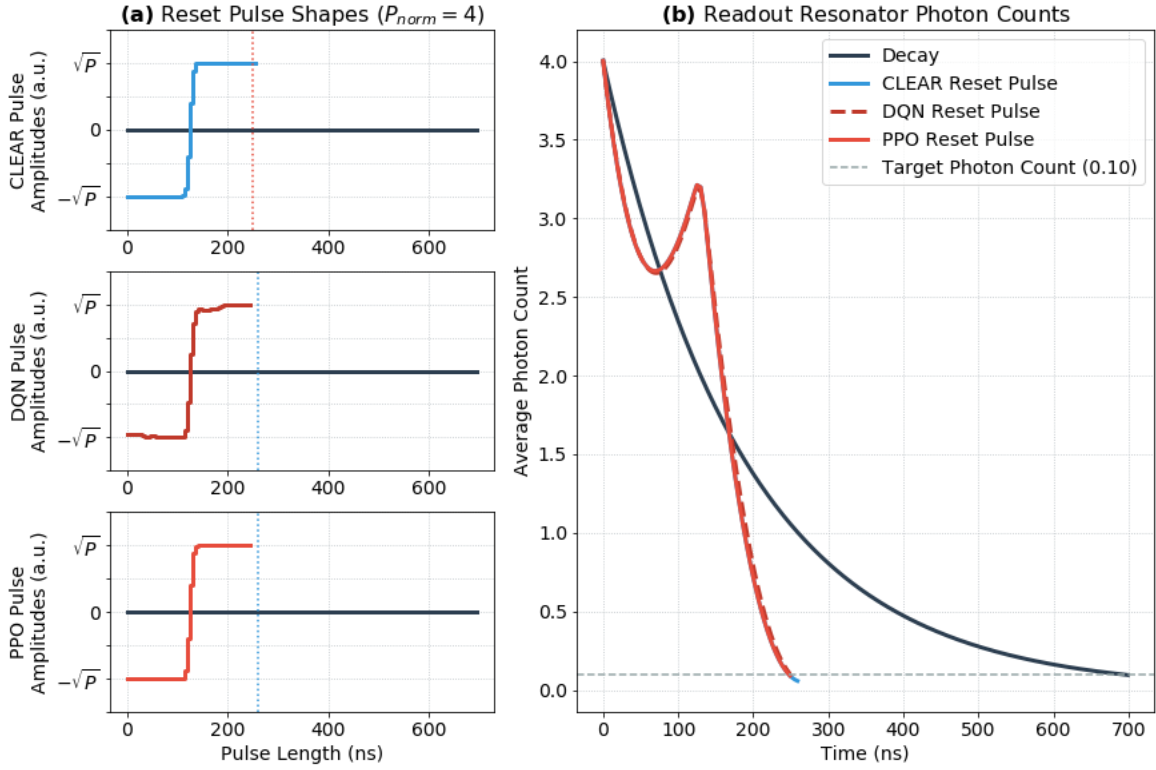


Figure 6-1: (a) Generated CLEAR, DQN and PPO pulse shapes for single qubit readout resonator reset. In this case, the conventional rectangular readout pulse passively waits for the photons to decay from the resonator, thus the pulse amplitude is 0. (b) Photon counts over time due to the generated pulse shapes.

### 6.3.2 Comparing DQN and PPO Optimizations

While the pulses generated by DQN and PPO are largely identical and allow us to reach our target photon count in equal time, there are some stark differences in the training cycles of the two methods. Particularly, we see that DQN training cycles are much slower and much more inefficient compared to PPO training cycles. For our reported 250 ns reset pulses, it took DQN about 79,000 training episodes (or about 2 million training steps) to find an optimal pulse. PPO on the other hand only required about 4,000 single-step training episodes, proving to be a much more sample-efficient approach.

These differences are illustrated in Fig. 6-2. Here we represent the different pulses generated using t-SNE, a method that reduces high dimensional data to a two-dimensional space where similar high dimensional points are spatially closer together in the reduced dimensional space [50]. We first can look at the top row of Fig. 6-2, where a marker's, or pulse shape's, color is determined by the number of remaining photons. Here the DQN algorithm (top-left) needs to explore a wide range of possible pulse shapes, often producing similar pulse shapes many times. This isn't surprising as the DQN algorithm employs a greedy exploration strategy that forces the algorithm to choose random amplitudes at a decreasing frequency. Unfortunately, this is the smallest amount of exploration and training episodes we found that could produce this optimal pulse shape. The PPO algorithm (top-right) on the other hand does not require random exploration and is much more sample efficient, where it spends about half of its training episodes fine-tuning good pulse shapes compared to exploring poor-performing pulse shapes.

In addition to the neural network scaling issues discussed in Chapter 5 Section 3.1, DQN provides no advantage in pulse length minimization compared to PPO (it actually performs worse in Chapter 7 Section 3's single qubit injection setup) and is considerably slower to train. Thus, we do not try to employ DQN for future multi-qubit environments and work solely with CLEAR pulses and PPO generated pulses.

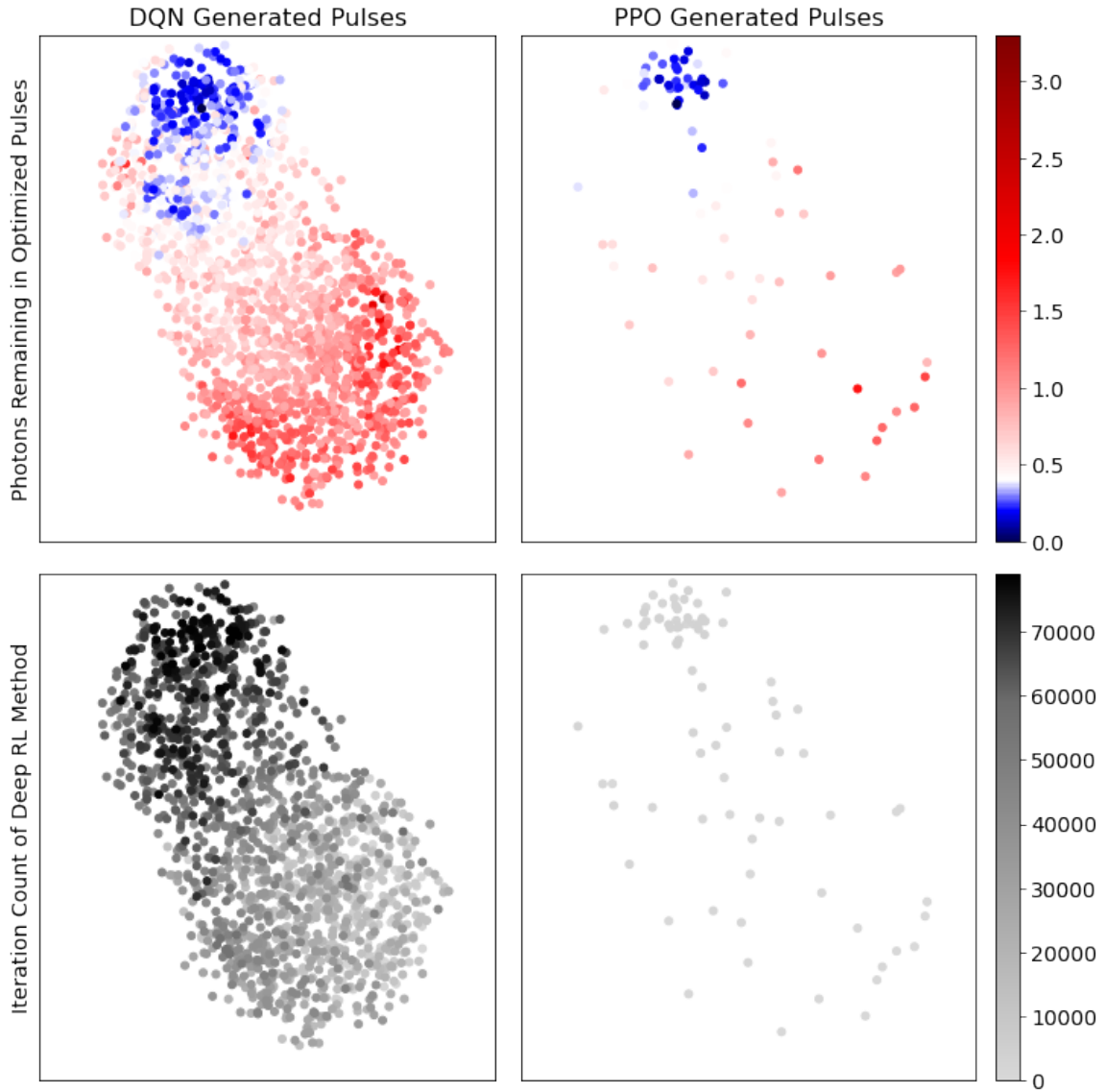


Figure 6-2: 2 dimensional representation of 25 dimensional pulses generated by DQN and PPO using t-SNE. With t-SNE, similar high dimensional data points are situated close together in the 2 dimensional representation. On the top row, each marker (pulse) is colored based on its reward, or number of photons remaining while on the bottom row each marker is colored based on the iteration it was made in its optimization cycle. Here we show every 50th pulse generated.

## 6.4 5 Qubit Simulation

### 6.4.1 Results

Following our single-qubit readout reset pulse optimization, we expand our work to generate optimal pulse shapes for a simulated environment with 5 qubits (Tab. 3.1). We again use the conventional passive reset setup as our base model and look to improve upon it using a CLEAR pulse backed with Bayesian optimization, as well as a PPO generated pulse. As discussed in Chapter 3 Section 3 and 5, expanding to a multiplexed setup introduces various forms of cross talk that can make ideal pulse shapes more complex and harder to optimize.

For our 5 qubit multiplexed readout setup, a passive reset setup requires waiting for 700 ns for all resonators to have photon counts below our target of a photon number of 0.10. We find that resonator reset pulses generated by PPO are minimized to a length of 380 ns. For CLEAR pulses backed by Bayesian optimization, the minimum pulse length is 460 ns. Compared to the 700 ns required for the conventional passive photon decay method, this constitutes a 45.7% ( $\sim 2.18 * t_{k,avg}$ ) time decrease for PPO and a 34.3% ( $\sim 1.64 * t_{k,avg}$ ) time decrease for CLEAR respectfully.

The pulse shapes generated by the various methods can be seen in Fig. 6-3(a). First, looking at the generated CLEAR pulses, we see that they are following a similar pattern as in the single-qubit reset optimization where we have a negative amplitude followed by a positive one. However in this multi-qubit environment, the amplitudes no longer are at the most negative and positive values of the available amplitude range. This can be attributed to two facts. First, in the multi-qubit setup all reset pulses are the same length, largely dependent on the individual resonator that needs the most time to reset. This means that the pulses for the other resonators need to find pulses for a specific pulse length that isn't necessarily its minimum, and thus amplitudes can be smaller in magnitude (see Section 4.2 for more details). Second, the multi-qubit environment also introduces cross-talk from off resonant pulses and from coupling effects between resonators and non-paired qubits, which both affect what the optimal pulse shapes.

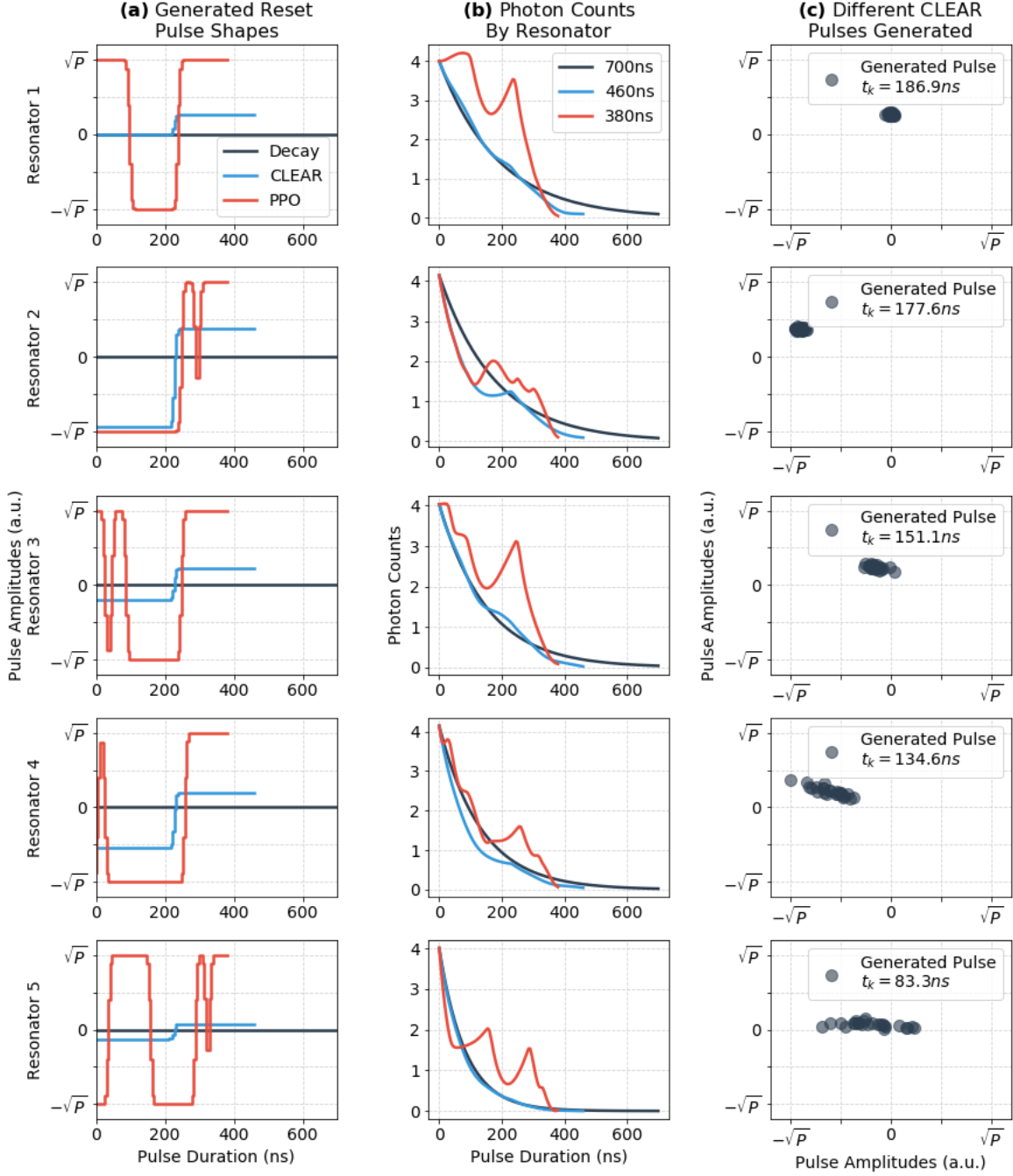


Figure 6-3: (a) Generated CLEAR, DQN, and PPO pulse shapes for single qubit readout resonator photon injection and stabilization. In this case, the conventional rectangular readout pulse passively waits for the photons to decay from the resonator, thus no pulse amplitude is applied. (b) Photon counts over time caused from the resulting generated pulse shapes. (c) Distribution of 25 generated CLEAR reset pulses.

Moving to the PPO generated pulses, we see that these pulses are more complex compared to CLEAR pulses. The PPO generated pulse amplitudes gravitate to the highest and lowest available amplitudes, fluctuating between the two at various frequencies. We can also observe the photon count overtime per resonator with the various reset pulses applied in Fig. 6-3(b). Here we see that similar to a passive reset, the CLEAR pulse photon counts mostly tend to strictly decrease over time. PPO pulse photon counts on the other hand fluctuate between decreasing and increasing frequently, likely an artifact of certain resonator states being easier to remove large amounts of photons from, even if that requires slightly increasing the photon count to get to those states.

### 6.4.2 Flexibility in Generated Pulse Shapes

One common occurrence we start seeing in multi-qubit pulse shaping optimizations (in the context of both reset and injection) is the effect of the resonator’s photon decay time  $t_k$  on the optimized pulse shapes. More specifically, for resonators with lower photon decay times, the range of pulses that can fulfill the reset goal expands as shown in Fig. 6-3(c). This isn’t too surprising given our system has resonators with varying  $\kappa$  and  $t_k$  values, and those with lower  $t_k$  will naturally decay quicker. Because we require all 5 pulses to be equal in length in our optimization setup, this means some individual pulses are not necessarily optimal (particularly pulses for resonators 4 and 5) as they could potentially be quicker compared to the other pulses. However, the overall system is still at its minimum required pulse length.

## 6.5 Summary

In this chapter, we compared optimized active readout resonator reset pulse shapes compared to passively turning off a readout reset pulse and letting photons naturally decay from the resonators. In a single-qubit environment, we found deep reinforcement learning methods, DQN and PPO, produced largely identical pulse shapes to the 2 amplitude-segment CLEAR reset pulse. These optimized reset pulses were 250-

260 ns long compared to passive reset which required 700 ns. The single-qubit reset optimization cycles revealed that DQN compared unfavorably to PPO as it required much more training and optimization space exploration compared to PPO to generate pulse shapes of the same length. Next, we looked at multi-qubit readout resonator reset. Here we found that PPO generated more complex pulse shapes compared to the CLEAR reset pulse shapes. The PPO generated pulse shapes also outperformed the CLEAR reset pulse shapes, requiring 380 ns compared to 460 ns. Both optimized pulses were substantial improvements to the passive photon decay method that required 700 ns.

# Chapter 7

## Resonator Photon Injection

### 7.1 Introduction

In the second step of multiplexed superconducting qubit readout, we look at “photon injection,” or the process of populating and stabilizing the readout resonators with photons. Here, we are concerned about generating a readout pulse shape that can allow us to probe the dispersive shifts as quickly as possible. The faster photons can be injected and stabilized in the resonators, the faster we can probe the dispersive shift. With the conventional rectangular pulse, we typically see the resonator photon counts oscillate and reach a steady-state based upon the normalized drive power  $P_{\text{norm}}$  and any cross-talk interactions that may be encountered in a multiplexed readout system. Fig. 7-1(b) shows the photon counts of our simulated five-qubit system with  $P_{\text{norm}} = 4$ . As with resonator reset pulse shaping, resonators with lower photon decay times (e.g., resonator 5) typically reach a steady photon count quicker than other resonators. Here we will still enforce that all pulse shapes for a given single- or multi-qubit system are equal length to each other and aim to optimize for pulse shapes that shorten overall the time needed to steady those photon counts as much as possible.

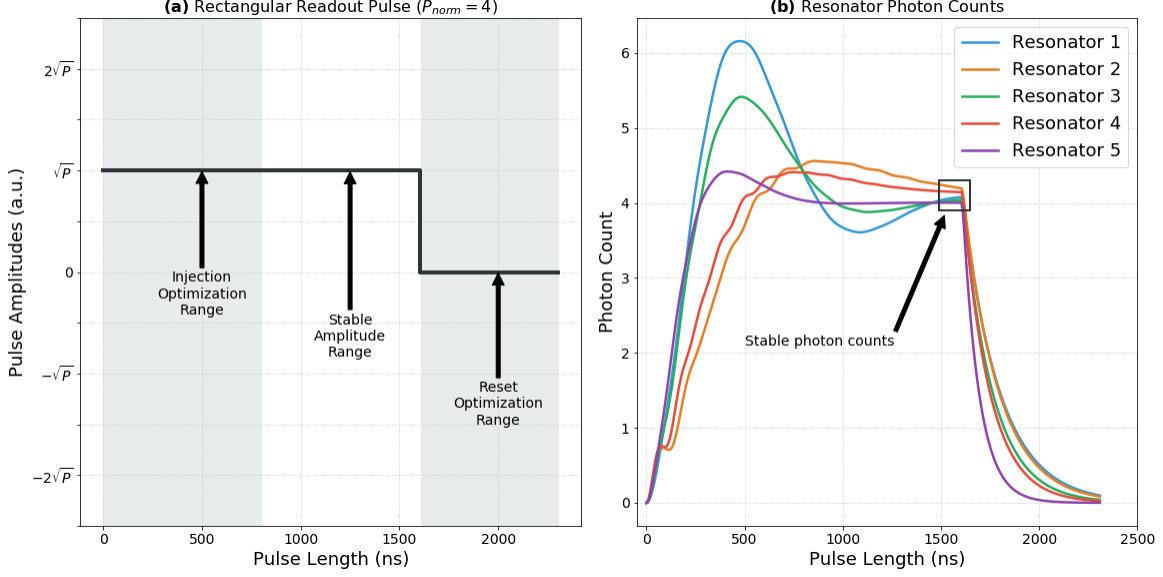


Figure 7-1: A basic rectangular readout pulse at a normalized drive power  $P_{\text{norm}} = 4$  (a) and the resulting photon counts for the resonators in a 5-qubit system (b). There are three main sections of the readout pulse that we consider. The first section is the photon injection optimization range at the beginning of the pulse, which we actively aim to minimize the required length of. Second, the middle section is a constant amplitude section that is driven at  $\sqrt{P}$  and is equal in length to the preceding pulse section that is being length optimized. Finally the last segment is reset optimization section that was discussed in Chapter 6. As we seen in (b), the photon counts from a rectangular pulse have an oscillating evolution where the photon counts eventually stabilize based on the normalized drive power  $P_{\text{norm}}$ .

## 7.2 Optimization Setup

Similar to our resonator reset optimization setup, we require the optimized pulses to be unconditional on the underlying qubit states. We still allow a set of 1,024 amplitudes available to be selected, however, the minimum and maximum amplitudes are now 0 and  $2\sqrt{P}$  respectfully. One major concern we now have with resonator photon injection, especially in simulated environments, is ensuring we do not exceed the critical photon numbers that cause the dispersive Jaynes-Cummings Hamiltonian approximation (Eq. 3.4) to break down. To combat this in a simulated environment, we introduce a tunable penalty term  $\Phi$  that is set to a large value ( $> 100$ ) to avoid pulses that inject too many photons in the resonators. The penalty is applied if any resonator's critical photon number is exceeded during the readout pulse.

An additional difference between the photon injection and reset pulses is that we do not optimize the entire photon injection pulse. This follows the CLEAR pulse setup, where we only optimize the beginning of the photon injection pulse to speed up the photon injections and stabilization [34]. For simplicity, we have the length of the optimized portion and length of the unoptimized portion of the photon injection pulse be equal as illustrated in Fig. 7-1(a).

In simulation, we cannot directly calculate how well a readout pulse shape will enable us to discriminate qubit states. Instead, we aim to find pulses that inject a stable number of photons determined by the normalized drive power. Reaching a stable photon count allows us to probe the separation in the resonator states (as depicted in Fig 7-5) based on the underlying qubit state. The conventional single-amplitude rectangular pulse already does this in the real world, but it is a slow process to get the photon counts and resonator states to stabilize. Thus to find optimal pulses, we set the reward for the simulated setup as the sum of the absolute difference between a resonator's photon count ( $n_r$ ) and its stable photon count determined by the normalized drive power ( $\hat{n}_r$ ). For a pulse of length  $L$ , we look for a stable photon count for the last  $L'$  nanoseconds of that pulse. Because minor oscillations can still occur, we define a stable photon count as one that is within 0.10 photons for the entire time window of  $L'$  nanoseconds. If this condition is not met, the reward is any penalty accrued plus the negative sum of the absolute differences between the photon counts and the stable photon counts for the last  $L'$  nanoseconds. The full reward function for the set of resonators  $R$  is defined below as:

$$\text{Reward} = \begin{cases} -\Phi, & \text{if } \forall r \in R, \forall l \in [L - L', L], |\hat{n}_r - n_r^l| \leq 0.1 \\ -\Phi - \sum_{r \in R} \sum_{l=L-L'}^L |\hat{n}_r - n_r^l|, & \text{otherwise.} \end{cases} \quad (7.1)$$

As in our reset pulse optimization, for CLEAR pulses generating using Bayesian optimization, we use 250 random initial points and up to 2,500 optimization points. Then again for PPO generated pulses, we allow up to 51,200 training episodes.

## 7.3 Single Qubit Simulation Results

We begin with optimizing single qubit system pulse shapes, comparing the conventional single-amplitude rectangular pulse to optimized CLEAR, DQN, and PPO generated pulses. Like with our resonator reset simulations, we use the qubit and resonator parameters from Qubit 1 in Tab. 3.1 and ignore all other qubits and resonators to create a single-qubit system. We set the stable photon count length,  $L'$ , to 100ns to minimize the required pulse length as much as possible.

Following optimization training cycles, we see the optimized pulses allow us to substantially decrease the necessary length required to reach a stable photon count (Fig. 7-2). The conventional single amplitude rectangular readout pulse requires 1470ns to reach a stable photon count. In our CLEAR and PPO pulse, we can minimize the required pulse length to 550ns, a 62.6% reduction (around 4.92 photon decay times). For DQN, the shortest pulse we were able to optimize for was 600ns, a 59.2% reduction (around 4.65 photon decay times).

In terms of the pulse shapes generated by the optimization methods, we see that similarly to the reset pulses, the DQN and PPO pulses seem to follow a similar pattern to the CLEAR pulse. Again, we largely see that the pulses optimize to the highest available amplitude in the first half of the optimizable range, while the second half amplitude is the lowest available amplitude. This again follows the idea that ideal way to shape these single-qubit readout pulses is to overshoot our target initially and follow up with compensating amplitude.

It is unclear why the DQN was unable to find optimal pulse shapes at lower pulse lengths. It is worth noting that the DQN pulse struggles to mirror the generated CLEAR pulse like the PPO generated pulse largely can. Similar to the discussion in Chapter 6 Section 3.2, the DQN pulse again requires an incredibly larger number of training episodes compared to PPO. Thus, paired with its reduced performance in shortening the optimized pulse length, we again do not use DQN when scaling to multi-qubit environments.

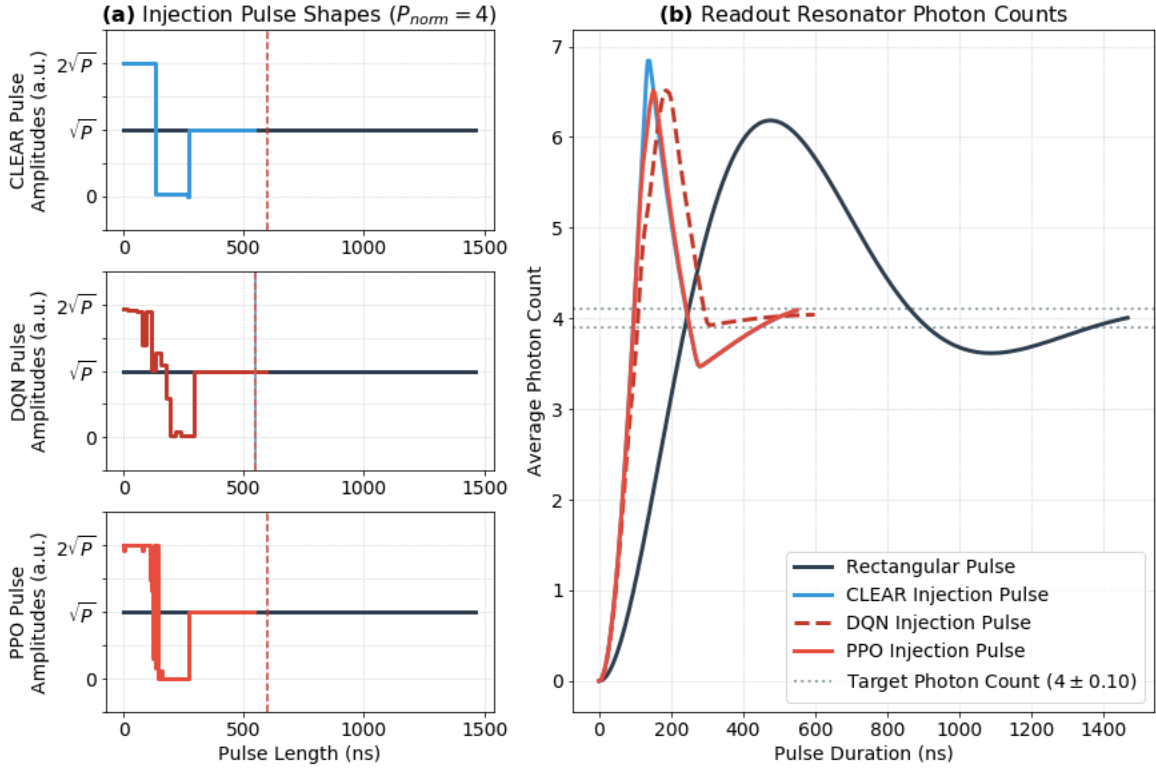


Figure 7-2: (a) Generated CLEAR, DQN, and PPO pulse shapes for single qubit readout resonator photon injection and stabilization. In this case, the conventional approach is a single amplitude rectangular pulse. (b) Photon counts over time as a result of applying the generated pulse shapes.

## 7.4 5 Qubit Simulation Results

### 7.4.1 Injection Results

For our 5-qubit multiplexed readout simulation, we use the system parameters from Tab. 3.1 and compare the conventional rectangular pulse approach, a CLEAR pulse optimized with Bayesian optimization, and finally a deep reinforcement learning approach using PPO. Similar to the single-qubit setup, We set the stable photon count length,  $L'$ , to 100ns.

The conventional set of single-amplitude rectangular readout pulses requires a pulse length of 1610ns to have all five resonator photon counts reach a steady target. Using a CLEAR pulse optimized with Bayesian optimization, we reduce the required pulse length to 780ns, a 51.6% reduction. Like we saw in multi-qubit reset, pulse shapes optimized using PPO produced either further pulse length reduction and only require a 600ns pulse length, a 62.7% reduction compared to a rectangular pulse.

The optimized CLEAR and PPO pulse shapes can be found in Fig. 7-3. With the optimized CLEAR pulse shapes, we see a similar pattern to what we saw with the multi-qubit CLEAR reset pulses. We continue to see a high amplitude segment followed by a lower amplitude segment, but the magnitudes of both amplitude segments are mostly not at the minimum and maximum of the available amplitude range for the various reasons discussed in Chapter 6 Section 4.1. With PPO generated pulses, we again see a similar pattern to the multi-qubit reset environment, where the optimized amplitudes largely stay at the amplitude range boundaries, and switch back and forth at various frequencies. What seems to be happening with the PPO generated pulse shapes here and in the multi-qubit reset environment is that it's still following the idea that we can overshoot our target and then compensate with the latter parts of the pulse shape. With these PPO generated pulse shapes, it seems to be doing that multiple times in the same pulse shape, allowing it to reduce the required pulse length compared to the multi-qubit CLEAR pulses.

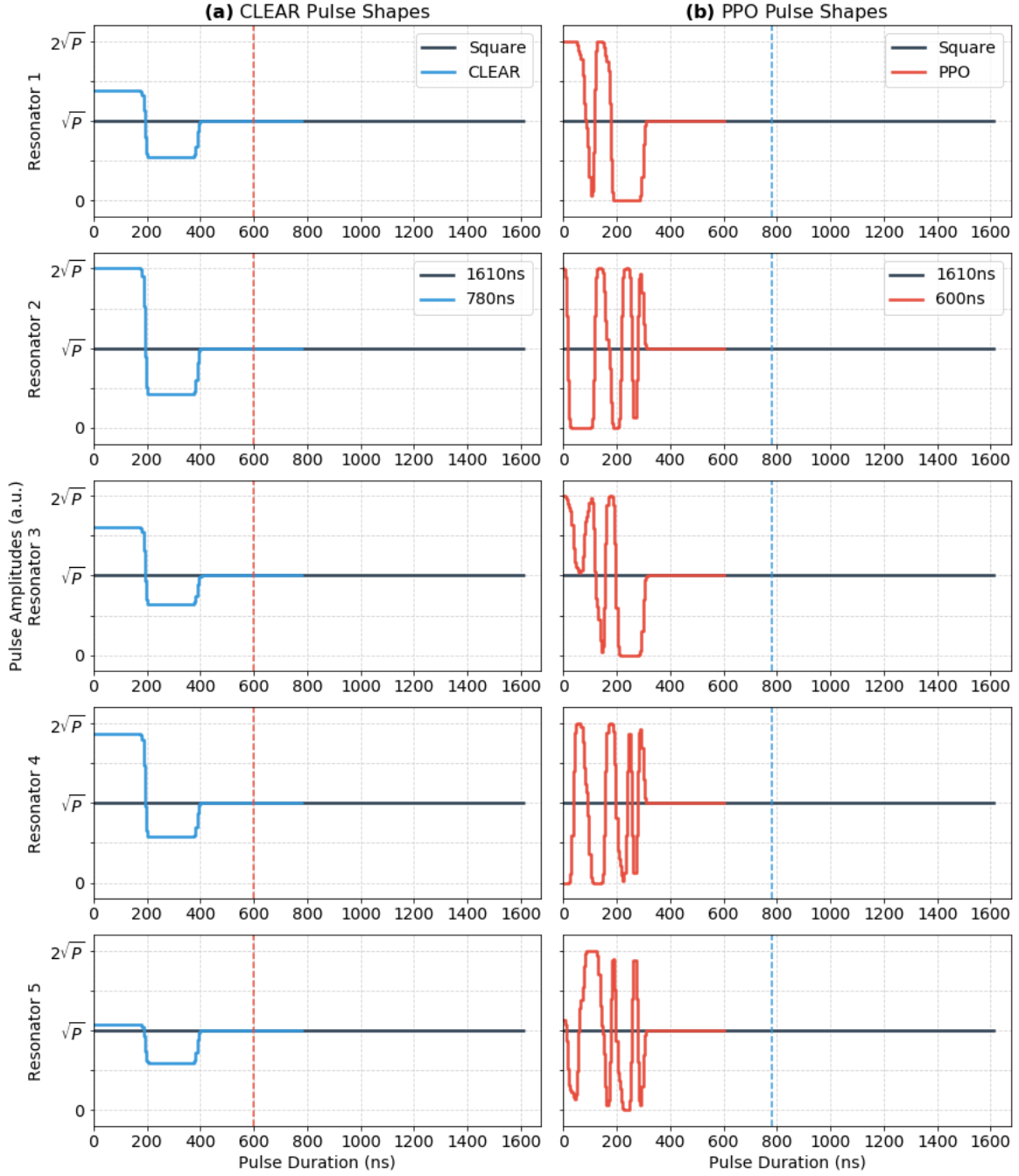


Figure 7-3: Generated pulse shapes for photon injection and stabilization using a rectangular pulse, (a) CLEAR pulse and (b) PPO pulse. In this setup, the amplitudes of the first half of the pulse are optimized while the second half of the pulse is at a constant amplitude  $\sqrt{P}$ .

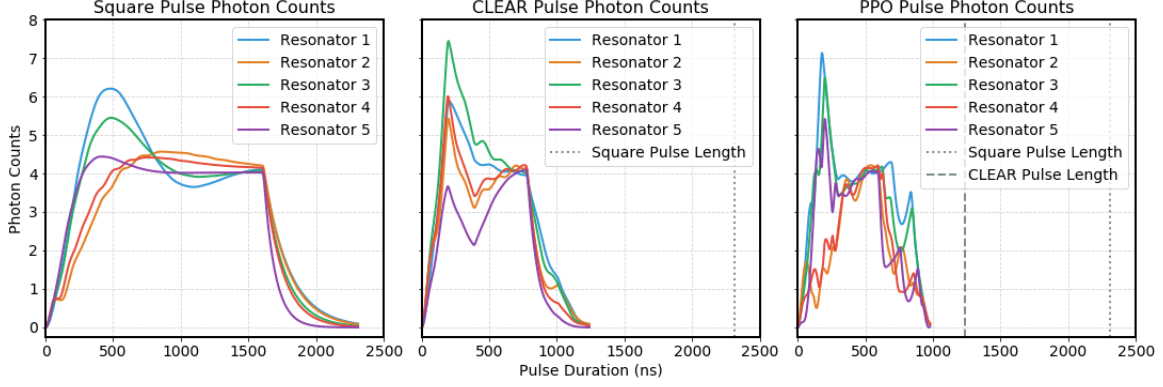


Figure 7-4: Resonator photon counts for the full readout pulse (photon injection and reset) using rectangular pulses (left), optimized CLEAR pulses (center), and pulses generated using PPO (right).

### 7.4.2 Entire Readout Pulse Results

Now, that we have optimized both readout reset and photon injection pulses, we can combine the two to generate the full readout pulse shapes. To recap, the conventional rectangular readout pulse followed by natural photon decay requires 2310ns, an optimized CLEAR pulse requires 1440ns, and an optimized PPO generated pulse requires 980ns.

The time-dependent photon counts of each applied pulse are illustrated in Fig. 7-4. All methods seem to rely on rapidly injecting photons and backing off before stabilizing with some of the resonators (1 and 3 in particular). While these photon spikes do occur when injecting the resonators with photons, it's important to remember we purposely optimize for pulses that do not allow for resonators to have their critical photon number exceeded.

More informative perhaps is observing the IQ-plane trajectories of the resonator states in Fig. 7-5. Here, we see that cavity states are separated based on the underlying qubit states. In our optimized CLEAR and PPO pulse shapes, we can approach those separated cavity states much quicker as shown by the increase in the marker separation, as well as take more direct trajectories to those steady states. While both the CLEAR pulse and PPO generated pulse IQ-planes highlight how much quicker the resonator state's are changing, we do also see that typically, PPO generated pulses

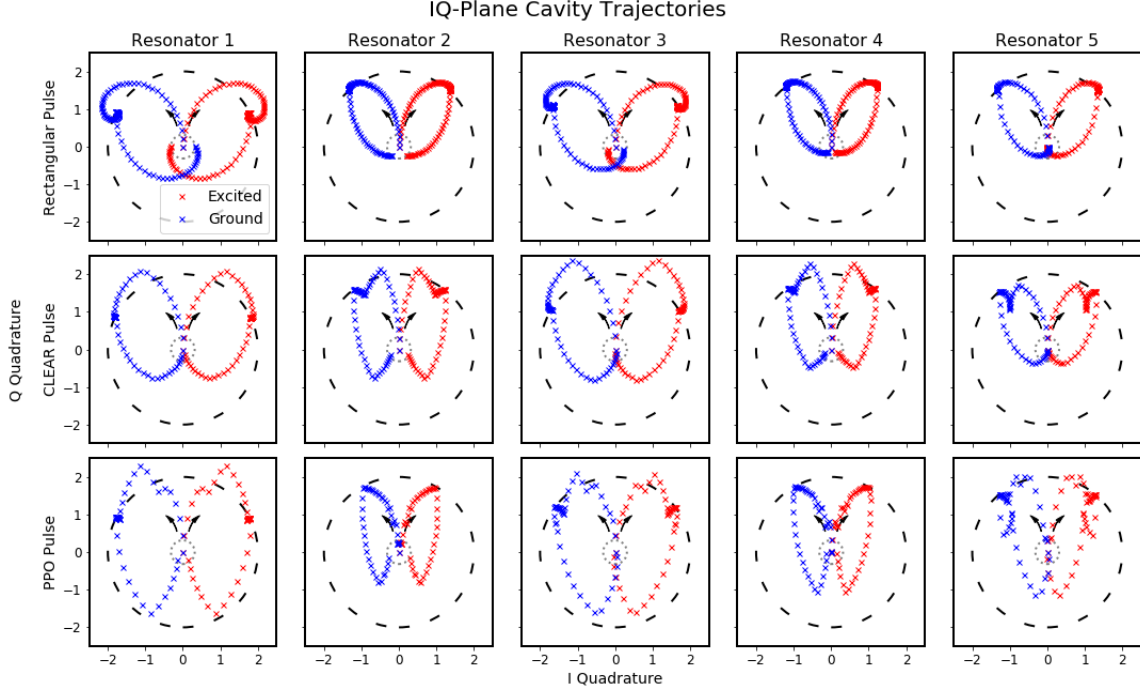


Figure 7-5: IQ-plane cavity trajectories per resonator for each type of generated full readout pulse (rectangular, CLEAR, PPO). Trajectories with red markers are associated with a system where all 5 qubits are in the excited state, and vice versa for with ground state trajectories with blue markers. Markers are spaced 20ns apart to highlight the speed in which cavity states are reached. Each IQ-plane also has a dashed grey circle that represents a resonator state with 4 photons (stabilization target), and a smaller dotted grey circle that represents a resonator state with 0.10 photons (reset target).

seem to be taking more direct paths than those from the CLEAR pulses (especially resonators 2 and 4).

One IQ-plane trajectory that stands out is for the PPO pulse in resonator 5 (bottom-right of Fig. 7-5). Here we see that unlike most other trajectories, this one does not have a very direct and efficient trajectory. The reasoning behind this is the same as we saw with the different CLEAR pulses being found for CLEAR reset pulses in Fig. 6-3(c). Because resonator 5 has a much lower photon decay time than the other resonators, it can naturally stabilize its photon count and naturally reset with conventional methods much quicker than the other resonators. When optimizing multiple pulses with the same length, this means there is likely to be a large number of possible pulses, many not entirely efficient, that allow us to reach our goal states.

One could try to optimize different lengths for each of the individual pulses, but the total length of the multiplexed readout pulse would likely still be the same as it's already dependent on the resonators with higher photon decay times.

## 7.5 Summary

In this chapter, we compared optimized readout pulse shapes for injecting photons into readout resonators as quickly as possible. We compared these optimized pulse shapes to the conventional method of using a single-amplitude rectangular pulse shape. In the single-qubit environment, we saw substantial pulse length reductions in the optimized pulse shapes compared to the rectangular pulse that required 1470 ns. The CLEAR and PPO generated pulse shapes, which were basically identical, produced 550 ns long pulse shapes while DQN produced a 600 ns pulse shape. Again, similar to the optimized reset pulses, DQN required a much more intense training cycle compared to PPO, even though in this case it produced a longer pulse shape. In the multi-qubit environment, we again dropped DQN and optimized a CLEAR and PPO generated pulse shape. Here, we again saw substantial pulse length reductions. While the rectangular pulse needed 1610 ns, the CLEAR pulse required 780 ns and the PPO generated pulse only required 600 ns.

# Chapter 8

## Experimental Setup

In this chapter, we will briefly discuss how the actual experiment will be set up. First, we will discuss the actual cryogenic hardware setup that will be used to conduct these pulse shaping optimization experiments. Then we will discuss the experimental designs for both optimizing ideal resonator reset pulses and readout resonator photon injection pulses on the actual hardware.

### 8.1 Measurement Setup

As discussed before, these experiments and simulations are performed using a superconducting 5 qubit multiplexed chip. In the experimental measurement setup, the qubit chip resides in the mixing chamber of a cryogenic dilution refrigerator that allows us to lower temperatures to around 10 mK. Qubit control and readout pulses are programmed and sent to an arbitrary waveform generator (AWG), whose output signal is up-scaled to the various qubit transition frequencies using an IQ-mixer and local oscillator. The control and readout pulses are then combined and carry on to the 5-qubit chip.

Once the readout signal acquires the qubit-state dependent phase shifts, we need to be able to measure that output signal. From the chip, as is seen in Fig. 8-1(a), a series of amplifiers boost the readout signal to various dilution refrigerator temperature stages. Once the signal reaches room temperature, the signal is once again

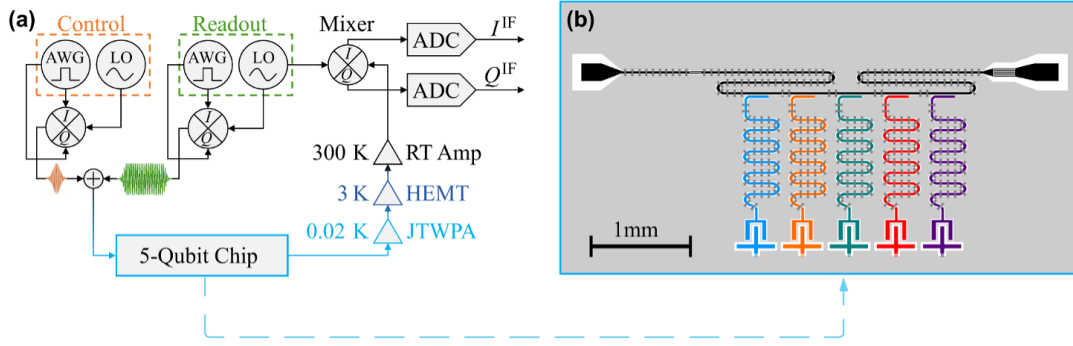


Figure 8-1: (a) Simplified setup for qubit control and multiplexed readout for a superconducting multi qubit system. On the left, pulse shapes for qubit control and readout are created, combined, and sent to the 5-qubit chip. On the right, after passing through the chip, the readout signal returns to room temperature after passing through a chain of amplifiers, allowing us to acquire measurement data. (b) A magnified view of the 5-qubit chip in the super cooled mixing chamber. The chip’s qubit’s are connected to a single drive line in a multiplexed fashion, allowing us to control and readout all 5 qubits with a single multiplexed signal.

amplified and then sent in a heterodyne detector that eventually allows the signal’s IQ components to be digitized and used to calculate the values used in our experimental reward functions.

## 8.2 Labber Pulse Optimization Driver

In our experimental setup, control and readout pulses are programmed in the Labber software program that interfaces with multiple arbitrary waveform generators [51]. To be able to perform readout pulse optimization in the various experiments, we created a custom Labber driver to construct an online optimization loop. This driver both sends in custom pulse shapes generated by the optimization software, as well as takes in the outputted readout data that forms our reward spaces described in the chapter’s upcoming sections. Importantly, the optimization software is the same as what is used in the simulated environment, as the overall software package is modular such that different environments, both simulated and experimental, can be easily switched in and out.

### 8.3 Resonator Reset Experiment Design

Like in our simulation environment we can optimize experimental resonator reset pulses by relying on calculating the number of photons in the resonator's following that optimized reset pulse. In the experimental setting, we use what is known as Ramsey experiment, which allows us to extract the number of remaining photons in a cavity following a pulse [52, 53]. As seen in Fig. 8-2, we apply a pulse sequence where we first use a control pulse to set various qubit states, followed by a readout pulse with or without an optimized reset portion. Following this, we apply a  $\pi/2$ -pulse, wait time  $t_R$ , and apply a second  $\pi/2$ -pulse. We perform this experiment for a range of  $t_R$  times. With this collected data, we can fit the following function,

$$S(t_R) = \frac{1}{2} [1 - \text{Im}\{\exp\{-(\Gamma_2 + i\Delta)t_R + i(\phi_0 - 2n_0\chi\tau)\}\}], \quad (8.1)$$

where  $\tau = (1 - e^{-\kappa + 2\chi i})/(\kappa + 2\chi i)$ ,  $\Gamma_2 = 1/T_{2E}$ , and  $n_0$  is the number of photons in the resonator at the start of the Ramsey experiment [34]. Given that we have all parameter values already from first characterizing our multi-qubit chip, we can fit the acquired trace data and solve for  $\phi_0$  and  $n_0$  in order to calculate the resonator photon count following a readout reset pulse.

### 8.4 Resonator Photon Injection Experiment Design

Following pulse shape optimization for readout resonator reset pulse shapes, we can then look to optimize experimental pulse shapes for injecting photons into the readout resonators. Here, we no longer need to worry about measuring actual photon numbers. Instead, we care about maximizing the qubit-state assignment fidelity  $\mathcal{F}_{\text{assignment}}$  while minimizing the readout pulse length.

The fidelity of qubit-state assignment is described as the probability of assigning a qubit state (ground or excited) correctly based upon how that qubit is explicitly

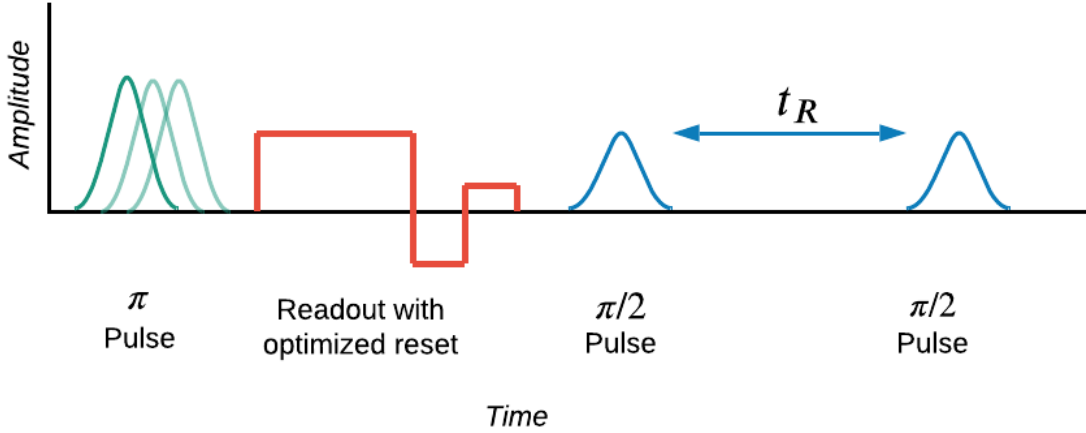


Figure 8-2: Sample Ramsey experiment pulse sequence. Here, a single Ramsey experiment begins by applying a set of  $\pi$ -pulses to initialize the qubit states. This is followed by a rectangular readout pulse that has an optimized reset component at the end to empty the resonator as quickly as possible. Following the readout pulse, we apply two sets of  $\pi/2$ -pulses, spaced apart by a time  $t_R$ . These  $\pi/2$ -pulses rotate the qubit-state dependent Bloch vector on the equator, and then back to the z-axis. During that time between the two  $\pi/2$ -pulses, various amounts of detuning occurs which affects how vector returns back to it's original vector position following the second  $\pi/2$  pulse.

prepared. For a single qubit  $i$ , the qubit-state assignment fidelity is

$$\mathcal{F}_i = 1 - [P(0_i|\pi_i) + P(1_i|\emptyset_i)]/2, \quad (8.2)$$

where  $\emptyset_i$  denotes preparing the qubit in the ground state and  $\pi_i$  is describing putting the qubit in the excited state, while  $0_i$  and  $1_i$  are the predicted qubit states. To expand this measure of fidelity to a multi-qubit system, we take the geometric mean of all the individual qubit-state assignment fidelities. In our 5-qubit state setup this, the overall qubit-state assignment fidelity is

$$\mathcal{F}_{\text{assignment}} = (\mathcal{F}_1\mathcal{F}_2\mathcal{F}_3\mathcal{F}_4\mathcal{F}_5)^{1/5}. \quad (8.3)$$

To obtain an optimized readout pulse's qubit-state assignment fidelity, we can simply follow the steps required to obtain that fidelity using a conventional multiplexed rectangular pulse. First, we send a readout pulse to the multi-qubit chip

that probes the various qubit-state dependent resonator phase shifts. Upon passing through numerous amplifiers and reaching room temperature, the readout signal is then fed into a heterodyne detector. In the heterodyne detector, the readout signals I and Q components are digitized, providing the data source that can subsequently be used to discriminate the underlying qubit states.

There are a few approaches we can take to performing qubit-state discrimination. A standard approach is to use a matched filter where we apply an integration kernel to the outputted readout signals. This integration kernel reduces the readout signals to a single value that is dependent on the signal phase, for which a simple discriminator can be used to determine the qubit state [54]. Other methods employed include support vector machine's (SVM's) that classify the demodulated readout signal, as well as deep neural networks that classify raw readout signal data [55, 22]. Regardless of the method, we start by training/fitting a discriminator using the conventional rectangular readout pulse first. Then we apply the trained/fitted discriminator to readout signal outputs created generated by the optimized readout pulse shapes, trying to maximize  $\mathcal{F}_{\text{assignment}}$ .



# Chapter 9

## Conclusion and Future Work

### 9.1 Conclusion

In this research thesis, we have demonstrated that deep reinforcement learning methods can be employed to optimize multiplexed superconducting qubit readout pulses (Table 9.1). We looked at a variety of optimization methods to improve upon the conventional approach of applying a single-amplitude rectangular pulse followed by a passive wait for photons to naturally decay from the resonator. These methods included combining Bayesian optimization with the Cavity-Level Excitation and Reset (CLEAR) pulse, as well as two deep reinforcement learning methods, Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO).

First, we examined and expanded upon the CLEAR pulse, a pulse shaping technique that adds two tunable amplitude segments to the beginning and to the end of a typical rectangular pulse. These segments help inject photons into the readout resonator quicker as well as actively reset those resonator's following a measurement's conclusion. Because the original research regarding the CLEAR pulse only looked at optimizing a single-qubit system, and thus a single readout pulse, we expanded upon that research to support multiplexed readout by introducing Bayesian optimization. This allowed us to tune amplitude segments pairs for all five pulses simultaneously and in a data-efficient manner. In single-qubit readout, the optimized CLEAR pulses provided a 62.7% pulse length decrease compared to the conventional rectangular

readout pulse. In our multi-qubit readout simulation system, the CLEAR pulses provided a 46.3% pulse length decrease.

Next, we added more fine-tuned control using deep reinforcement learning (DRL). In particular, we looked at two methods. Deep Q-Networks (DQN) uses deep neural networks to predict the value of selecting a specific amplitude from a discrete space. With these value approximations, we can construct pulse shapes one amplitude segment at a time. The second DRL method employed was Proximal Policy Optimization (PPO), a method that *directly* predicts the single best amplitude from a continuous range using deep neural networks. In our case, PPO was constructed such that it could predict an entire set of readout pulses in a single neural network pass, saving valuable training time.

For single-qubit readout, DQN generated pulses that reduced the required pulse length by 60.8%, while PPO generated pulse shapes reduced the required pulse length required by 63.1%. In general, we do see that the CLEAR pulse shapes and the PPO generated pulse shapes are very similar in the single-qubit environment. This isn't too surprising as the CLEAR pulse should be optimal in an environment free of noise and cross-talk from other pulses and qubit-resonator pairs. Additionally, we found that DQN training proved to be both slower than PPO and more sample inefficient. Paired with its reduced final performance, we did not employ DQN for the multi-qubit simulation environment. In that multi-qubit environment, PPO reduced the required pulse length by 57.6% compared to the rectangular pulse. Compared to the optimized CLEAR pulse, the PPO generated readout pulse provided an additional 21.0% pulse length reduction.

Finally, following the success of our readout pulse shaping optimization in a simulated multi-qubit readout environment, we discussed the experimental setup and designs that will be used to perform the real-world experiments by the group in the future.

Altogether, the successful results in the simulated environment show that state-of-the-art machine learning methods can be used for optimal control in superconducting quantum environments. Additionally, this work provides a framework for using deep

reinforcement learning, or any other to be discovered optimization technique, in a variety of simulated and experimental settings.

	Pulse Shaping Optimization Method			
	Rectangular + Decay	CLEAR	PPO	DQN
Single-Qubit Reset	700 ns	260 ns	250 ns	250 ns
Single-Qubit Injection	1470 ns	550 ns	550 ns	600 ns
Single-Qubit Complete Pulse	2170 ns	810 ns	800 ns	850 ns
Multi-Qubit Reset	700 ns	460 ns	380 ns	-
Multi-Qubit Injection	1610 ns	780 ns	600 ns	-
Multi-Qubit Complete Pulse	2310 ns	1240 ns	980 ns	-

Table 9.1: Final readout pulse lengths for the various conventional and optimization methods in this research. The single-qubit environment includes optimized CLEAR pulses and pulses generated with PPO and DQN. In the multi-qubit environment, we only compare optimized CLEAR pulses and pulses generated with PPO.

## 9.2 Future Work Possibilities

### 9.2.1 Additional Applications

The first broad set of possibilities for future work is using the software package to generate optimal pulse shapes for different use cases, both in simulated environments and experimental setups. Outside of multiplexed readout, a couple of areas of using deep reinforcement learning have been explored in simulated environments already. The most substantial work involves using AlphaZero to generate optimal single-qubit gate pulse shapes [11]. However the AlphaZero approach in that research suffered from similar issues that we found in non-PPO algorithms, namely that they were too slow training-wise and that existing optimization methods could sometimes outperform them. Based on what we found in this thesis, it would seem that a deep policy gradient algorithm like PPO could be more successful in a context like single- and two-qubit gate optimal pulse shaping. Importantly, our work provides a path toward being able

to implement these DRL optimal control approaches in experimental settings. To the best of our knowledge, this would be the first experimental implementation of using DRL in a quantum optimal control setting.

It’s important to note that software created for this research is not restricted to only working with the readout simulation environment we describe. Instead, it is a modular system designed to allow users to swap in different optimization techniques (CLEAR, PPO, DQN, etc.), as well as different simulation parameter values and entirely different environments. For those different simulation environments, the software package only requires a user to accept pulse shapes in an array-like format and then output a scalar reward signal. All simulation logic to get from a pulse shape to that reward signal is entirely up to an end-user to implement, making the software package extremely extendable. The same holds true for using the software system in experimental environments. Instead of implementing simulation software, one can send the generated pulse shapes to an instrument interface like Labber and implement a method to convert measurement data to a scalar reward, all while not needing to change any of the optimization software.

### 9.2.2 Pulse Length Optimization

One of the largest opportunities for extending the current deep reinforcement learning implementation is finding a way to include optimizing the length of the generated pulse in addition to the actual pulse shape. Currently, we have to set a specific, locked-in pulse length, as well as the length of each pulse segment, at the start of each new optimization cycle. If that optimization proves to be successful, we reduce the pulse length and keep this cycle going until we find a pulse length that can’t be shortened any further. Obviously, having to run multiple optimization cycles is a bottleneck especially for deep learning applications where training can take long amounts of time.

The goal of future work from a reinforcement perspective would be to include finding the optimal pulse length in addition to the optimal pulse amplitudes. To do this, we could create set pulse segment lengths, and look to include a “stop” segment

as part of the available action set in our reinforcement learning environment. One could even have a setup where individual pulses could be stopped at different times, eliminating strange individual pulses associated with resonators that have shorter photon decay times (see Fig. 6-3(c) and resonator 5 in Fig. 7-5). A reward signal would need to be carefully designed to incorporate both minimizing the pulse length and reaching the desired resonator state. While this would most likely increase the overall training time needed to perform a single optimization cycle, it would eliminate the need to run multiple optimization cycles and thus should reduce the overall time needed to find optimal readout pulses.

### 9.2.3 Continuous Experimental Optimization

Another opportunity for practical use of deep reinforcement learning in superconducting quantum computing systems is with recalibration efforts. It is common for systems to need frequent recalibration, whether it be changing  $T_1$  times, drift in the control fields used to drive logic gates or various sources of noise in the system or from the instruments controlling the system [56, 57, 58]. This means that if we use PPO to optimize an ideal set of readout pulses, those pulses may become less effective over time.

PPO provides an intriguing possibility to try and compensate for those systems changes in real-time. Once an optimal readout pulse is generated, one could theoretically still allow PPO to continuously optimize an ideal readout pulse shape. As discussed in Chapter 5 Section 3.2, when PPO is updating its neural network weights, it generates a set of policies (pulse shapes) from the existing policy distribution, then trains off the best performing new policies. If we were to perform this step whenever our pulse performance drops below a certain threshold, we could potentially “follow” the changes in the system as they are occurring, rather than shutting down an experiment and recalibrating from scratch.



# Bibliography

- [1] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, “A quantum engineer’s guide to superconducting qubits”, *Appl. Phys. Rev.*, 6(2), 2019.
- [2] R. Babbush et al. F. Arute, K. Arya, “Quantum supremacy using a programmable superconducting processor”, *Nature*, 574(7779):505–510, 2019.
- [3] J. Preskill, “Quantum Computing in the NISQ era and beyond”, *Quantum*, 2:79, 2018.
- [4] Alexandre Blais, Ren-Shou Huang, Andreas Wallraff, S. M. Girvin, and R. J. Schoelkopf, “Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation”, *Phys. Rev. A*, 69:062320, 2004.
- [5] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland, “Surface codes: Towards practical large-scale quantum computation”, *Phys. Rev. A*, 86:032324, 2012.
- [6] D.T. McClure, Hanhee Paik, L.S. Bishop, M. Steffen, Jerry M. Chow, and Jay M. Gambetta, “Rapid driven reset of a qubit readout resonator”, *Physical Review Applied*, 5(1), 2016.
- [7] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, A Bradford Book, Cambridge, MA, USA, 2018.
- [8] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panniershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis, “Mastering the game of go with deep neural networks and tree search”, *Nature*, 529:484–503, 2016.
- [9] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis, “Mastering chess and shogi by self-play with a general reinforcement learning algorithm”, *ArXiv*, abs/1712.01815, 2017.

- [10] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”, In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017.
- [11] J. J. Sorensen M. Dalgaard, F. Motzoi and J. Sherson, “Global optimization of quantum dynamics with AlphaZero deep exploration”, *npj Quantum Inf.*, 6:6, 2020.
- [12] Peter W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”, *SIAM Journal on Computing*, 26(5):1484–1509, 1997.
- [13] Richard P. Feynman, *Simulating Physics with Computers*, page 133–153, Perseus Books, USA, 1999.
- [14] I. M. Georgescu, S. Ashhab, and Franco Nori, “Quantum simulation”, *Rev. Mod. Phys.*, 86:153–185, 2014.
- [15] Michael Tinkham, *Introduction to Superconductivity*, Dover Publications, 2 edition, 2004.
- [16] V. Bouchiat, D. Vion, P. Joyez, D. Esteve, and M. H. Devoret, “Quantum Coherence with a Single Cooper Pair”, *Physica Scripta Volume T*, 76:165–170, 1998.
- [17] Yuriy Makhlin, Gerd Schön, and Alexander Shnirman, “Quantum-state engineering with josephson-junction devices”, *Reviews of Modern Physics*, 73(2):357–400, 2001.
- [18] G. Ithier, E. Collin, P. Joyez, P. J. Meeson, D. Vion, D. Esteve, F. Chiarello, A. Shnirman, Y. Makhlin, J. Schrieffer, and et al., “Decoherence in a superconducting quantum bit circuit”, *Physical Review B*, 72(13), 2005.
- [19] D. Vion, “Manipulating the quantum state of an electrical circuit”, *Science*, 296(5569):886–889, 2002.
- [20] 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, 2007.
- [21] M.D. Hutchings, J.B. Hertzberg, Y. Liu, N.T. Bronn, G.A. Keefe, Markus Brink, Jerry M. Chow, and B.L.T. Plourde, “Tunable superconducting qubits with flux-independent coherence”, *Physical Review Applied*, 8(4), 2017.
- [22] B. Lienhard, A. Vepsäläinen, L. C. G. Govia, C. R. Hoffer, J. Qiu, D. Riste, M. Ware, D. Kim, R. Winik, A. Melville, B. Niedzielski, J. Yoder, G. J. Ribeill, T. A. Ohki, H. K. Krovi, T. P. Orlando, S. Gustavsson, and W. D. Oliver, “Deep

neural network discrimination of multiplexed superconducting qubit states”, Unpublished, 2021.

- [23] Alexandre Blais, Ren Shou Huang, Andreas Wallraff, S. M. Girvin, and R. J. Schoelkopf, “Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation”, *Phys. Rev. A - At. Mol. Opt. Phys.*, 69(6):1–14, 2004.
- [24] A. Wallraff, D. I. Schuster, A. Blais, L. Frunzio, R.-S. Huang, J. Majer, S. Kumar, S. M. Girvin, and R. J. Schoelkopf, “Strong coupling of a single photon to a superconducting qubit using circuit quantum electrodynamics”, *Nature*, 431(7005):162–167, 2004.
- [25] Alexandre Blais, Ren-Shou Huang, Andreas Wallraff, S. M. Girvin, and R. J. Schoelkopf, “Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation”, *Physical Review A*, 69(6), 2004.
- [26] J Dalibard, J.-M Raimond, and Jean Zinn-Justin, *Systemes fondamentaux en optique quantique = Fundamental systems in quantum optics / edite par J. Dalibard, J.-M. Raimond et J. Zinn-Justin.*, North-Holland, Amsterdam, 1992.
- [27] J.R. Johansson, P.D. Nation, and Franco Nori, “Qutip: An open-source python framework for the dynamics of open quantum systems”, *Computer Physics Communications*, 183(8):1760–1772, 2012.
- [28] F. Bloch, “Nuclear induction”, *Phys. Rev.*, 70:460–474, 1946.
- [29] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei, “Language models are few-shot learners”, 2020.
- [30] Richard Bellman, “The theory of dynamic programming”, *Bull. Amer. Math. Soc.*, 60(6):503–515, 1954.
- [31] Warren S. McCulloch and Walter Pitts, *A Logical Calculus of the Ideas Immanent in Nervous Activity*, page 15–27, MIT Press, Cambridge, MA, USA, 1988.
- [32] Sergey Ioffe and Christian Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, 2015.
- [33] David E. Rumelhart and David Zipser, *Feature Discovery by Competitive Learning*, page 205–242, Ablex Publishing Corp., USA, 1988.

- [34] D. T. McClure, Hanhee Paik, L. S. Bishop, M. Steffen, Jerry M. Chow, and Jay M. Gambetta, “Rapid driven reset of a qubit readout resonator”, *Phys. Rev. Applied*, 5:011001, 2016.
- [35] Johannes Heinsoo, Christian Kraglund Andersen, Ants Remm, Sebastian Krinner, Theodore Walter, Yves Salathé, Simone Gasparinetti, Jean Claude Besse, Anton Potočník, Andreas Wallraff, and Christopher Eichler, “Rapid High-fidelity Multiplexed Readout of Superconducting Qubits”, *Phys. Rev. Appl.*, 10(3):1–14, 2018.
- [36] N T Bronn, B Abdo, K Inoue, S Lekuch, A D Córcoles, J B Hertzberg, M Takita, L S Bishop, J M Gambetta, and J M Chow, “Fast, high-fidelity readout of multiple qubits”, *Journal of Physics: Conference Series*, 834:012003, 2017.
- [37] C. Macklin, K. O’Brien, D. Hover, M. E. Schwartz, V. Bolkhovskiy, X. Zhang, W. D. Oliver, and I. Siddiqi, “A near-quantum-limited Josephson traveling-wave parametric amplifier”, *Science*, 350(6258):307–310, 2015.
- [38] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger, “Gaussian process bandits without regret: An experimental design approach”, *CoRR*, abs/0912.3995, 2009.
- [39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis, “Human-level control through deep reinforcement learning”, *Nature*, 518(7540):529–533, 2015.
- [40] Christopher J. C. H. Watkins and Peter Dayan, “Q-learning”, *Machine Learning*, 8(3):279–292, 1992.
- [41] Navin Khaneja, Timo Reiss, Cindie Kehlet, Thomas Schulte-Herbrüggen, 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.
- [42] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, “Prioritized experience replay”, *CoRR*, abs/1511.05952, 2015.
- [43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, “Proximal policy optimization algorithms”, 2017.
- [44] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel, “Trust region policy optimization”, 2017.
- [45] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, “Continuous control with deep reinforcement learning”, *CoRR*, abs/1509.02971, 2015.

- [46] F. Motzoi, J. M. Gambetta, S. T. Merkel, and F. K. Wilhelm, “Optimal control methods for rapidly time-varying hamiltonians”, *Physical Review A*, 84(2), 2011.
- [47] Samuel Boutin, Christian Kraglund Andersen, Jayameenakshi Venkatraman, Andrew J. Ferris, and Alexandre Blais, “Resonator reset in circuit QED by optimal control for large open quantum systems”, *Phys. Rev. A - At. Mol. Opt. Phys.*, 96:1–11, 2017.
- [48] Yanbing Liu, Srikanth J. Srinivasan, D. Hover, Shaojiang Zhu, R. McDermott, and A. A. Houck, “High fidelity readout of a transmon qubit using a superconducting low-inductance undulatory galvanometer microwave amplifier”, *New Journal of Physics*, 16(11):113008, 2014.
- [49] 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, 2014.
- [50] Laurens van der Maaten and Geoffrey Hinton, “Visualizing data using t-sne”, *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.
- [51] Labber, “Labber is a general-purpose instrument control and lab automation software package, with particular focus on quantum applications.”, <https://www.keysight.com/us/en/products/software/application-sw/labber-software.html>.
- [52] Norman F. Ramsey, “A molecular beam resonance method with separated oscillating fields”, *Phys. Rev.*, 78:695–699, 1950.
- [53] E. L. Hahn, “Spin echoes”, *Phys. Rev.*, 80:580–594, 1950.
- [54] Colm A. Ryan, Blake R. Johnson, Jay M. Gambetta, Jerry M. Chow, Marcus P. Da Silva, Oliver E. Dial, and Thomas A. Ohki, “Tomography via correlation of noisy measurement records”, *Phys. Rev. A - At. Mol. Opt. Phys.*, 91(2):1–7, 2015.
- [55] Easwar Magesan, Jay M. Gambetta, A.D. Còrcoles, and Jerry M. Chow, “Machine Learning for Discriminating Quantum Measurement Trajectories and Improving Readout”, *Phys. Rev. Lett.*, 114:200501, 2015.
- [56] P. V. Klimov, J. Kelly, Z. Chen, M. Neeley, A. Megrant, B. Burkett, R. Barends, K. Arya, B. Chiaro, Yu Chen, A. Dunsworth, A. Fowler, B. Foxen, C. Gidney, M. Giustina, R. Graff, T. Huang, E. Jeffrey, Erik Lucero, J. Y. Mutus, O. Naaman, C. Neill, C. Quintana, P. Roushan, Daniel Sank, A. Vainsencher, J. Wenner, T. C. White, S. Boixo, R. Babbush, V. N. Smelyanskiy, H. Neven, and John M. Martinis, “Fluctuations of energy-relaxation times in superconducting qubits”, *Phys. Rev. Lett.*, 121:090502, 2018.

- [57] M. A. Fogarty, M. Veldhorst, R. Harper, C. H. Yang, S. D. Bartlett, S. T. Flammia, and A. S. Dzurak, “Nonexponential fidelity decay in randomized benchmarking with low-frequency noise”, *Physical Review A*, 92(2), 2015.
- [58] Jonathan J. Burnett, Andreas Bengtsson, Marco Scigliuzzo, David Niepce, Marina Kudra, Per Delsing, and Jonas Bylander, “Decoherence benchmarking of superconducting qubits”, *npj Quantum Information*, 5(1), 2019.