Machine Learning Optimization for Photon Detection in Trapped Ion Simulation

Paul Grajzl

Advisor: Professor David Hanneke
May 8, 2025

Submitted to the
Department of Physics & Astronomy of Amherst College
in partial fulfilment of the
requirements for the degree of
Bachelors of Arts with honors

Abstract

Trapped ion systems have emerged as a cornerstone of modern experimental physics, enabling breakthroughs in fields ranging from quantum computing and atomic clocks to precision spectroscopy, metrology, and molecular reaction dynamics. Their utility stems from the exceptional coherence, stability, and reproducibility of trapped ions, which enable non-destructive, high-resolution interrogation of quantum systems. For these reasons, there is a growing need for frameworks that not only simulate trapped ion behavior, but also intelligently guide experimental design. In this context, machine learning offers an invaluable tool — providing a powerful and flexible means of navigating complex parameter spaces, modeling nonlinear behavior, and predicting high-performing experimental configurations from data. A particularly valuable application of this platform is the study of molecular ion dissociation — specifically, the detection of dissociation of O_2^+ from a prepared rovibrational state using Laser-Cooled Fluorescence Mass Spectrometry (LCF-MS). While LCF-MS is a conceptually straightforward and widely used technique, the interplay between species in three-dimensional Coulomb crystals gives rise to rich and sensitive dynamics. In practice, these complexities make it difficult to consistently resolve dissociation events with high precision. To address this, we turn to numerical simulation and machine learning as tools to better understand and optimize system behavior — with the ultimate goal of improving our ability to quantify the number of O_2^+ ions in the trap from fluorescence signal alone. In this thesis, I leverage a high-fidelity simulation engine, QLICS, to generate large-scale datasets of across varied trap configurations. First, I construct a sample composite cost function to label outcomes and identify benchmark cases of strong number resolution. I then apply ensemble classification machine learning methods with carefully tuned hyperparameters to build a predictive model capable of guiding experimenters toward high-performing, robust configurations. Simultaneously, I use Monte Carlo sampling to capture stochastic variability and produce probabilistic estimates. I also integrate a novel interpolation technique with K-nearest neighbor scoring to yield a stability metric that quantifies local robustness in parameter space. Together, these tools not only support a principled, data-driven optimization framework, but also uncover a key physical insight: no single configuration performs optimally across all dissociation steps. This motivates a mixed strategy approach, in which distinct configurations are used for different numbers of dissociated ions to maximize signal resolution and experimental reliability.

Acknowledgments

I would like to thank Professor Hanneke for his consistent support and guidance throughout my four years at Amherst. I am also deeply grateful to Michael Mitchell for his mentorship and insight, and to Doug Hall, whose assistance with the HPC system proved invaluable.

This thesis is based upon work supported by the Amherst College Provost and Dean of the Faculty and by the National Science Foundation under grant PHY-2207623 (RUI PM). It was performed in part using high-performance computing equipment at Amherst College obtained under National Science Foundation grant CNS-2117377. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Imagine being able to onboard a new lab partner — one who doesn't need a full theoretical derivation every time, never forgets a result, and can instantly flag whether a new configuration is likely to succeed. A partner who can comb through thousands of simulations in seconds, spot patterns that no human ever would, and adapt its judgment with every new datapoint — all without getting tired, biased, or overwhelmed. Not someone to replace physical intuition, but something to amplify it — a model that learns directly from the physics, and helps guide the next move with precision and speed. This idea sits at the heart of this thesis and captures what machine learning makes possible in a setting where intuition alone falls short.

At its core, this project is about learning how to identify "good" experiments without having to stumble through dozens of bad ones to get there. There's a great deal of physics involved — we're trapping ions in a vacuum, driving dissociative transitions with UV lasers, and using photon count data to infer the composition of an ion crystal — but what ties the entire effort together is a single optimization problem: how do we efficiently explore a massive, high-dimensional space of experimental parameters in search of configurations that

actually work? And beyond that, how do we develop a model that can help us navigate that space in a systematic, generalizable, and semi-automated way?

At a broader level, the scientific motivation behind this work is to test whether fundamental constants — in particular, the proton-to-electron mass ratio, μ — remain invariant over time. While the Standard Model asserts that μ should be constant, several theoretical extensions, including models derived from string theory, allow for the possibility of slow temporal drift [1]. Observing even a small variation in μ would offer evidence for physics beyond the Standard Model, making it a compelling target for high-precision experimental efforts [1]. Our lab's approach centers on molecular oxygen ions (O_2^+), whose vibrational transition frequencies are highly sensitive to μ [2]. By preparing O_2^+ in specific quantum states and monitoring how those states evolve and dissociate under controlled experimental conditions, we aim to detect subtle shifts in transition energies that would indicate a change in μ over time [1, 2].

Achieving that level of sensitivity, however, depends critically on our ability to resolve the outcomes of those dissociation events with high precision. Specifically, we need to measure — cleanly and reproducibly — how many O^+ ions are produced when a dissociation event occurs. This count is the primary signal from which we infer vibrational transition energies [3]. Since our detection scheme only directly observes fluorescence from Be^+ ions, we rely on changes in motional coupling to indirectly sense the presence of dissociation products [3, 4]. By correlating the appearance of O^+ ions with shifts in the fluorescence signal, we can back out the vibrational states responsible for the dissociation, making precise ion counting essential for spectroscopy. If that number is ambiguous or buried in noise, the precision of the transition frequency measurement degrades, and with it, our ability to place meaningful bounds on μ variation. Thus, the focus of this thesis is to develop a robust, general-purpose framework for identifying dissociation events and optimizing trap configurations to enhance number resolution — that is, to distinguish between different numbers of O^+ ions in the trap with maximal clarity. By improving this resolution, we directly improve the quality of our

measurements and increase the sensitivity of the overall system to potential variations in μ. This work is therefore a foundational step toward enabling the broader scientific objective.

1.2 Why Use Machine Learning?

So why not simply repeat the experiment until a clear result emerges? In practice, it's not that straightforward. Even under ideal conditions, replicating the same experiment isn't trivial. Equipment drifts, crystals destabilize, detection optics shift out of alignment, and parameters thought to be stable can unexpectedly vary. In simulations, these issues are less severe, but the sheer size of the parameter space presents its own challenge. The knobs we're adjusting include laser detuning, beam geometries, trap voltages, ramp and modulation sequences, timing parameters, and more — a blend of continuous, discrete, and categorical variables with nonlinear and often interdependent effects. Brute-force search becomes unmanageable almost immediately. What we need is a way to explore that space efficiently and intelligently. This is where machine learning enters the picture.

Machine learning brings two essential capabilities to the table: it enables the system to be trained, and it makes the results reproducible. Once we've run a set of simulations and labeled the outcomes as "good" or "bad" based on consistent features in the dissociation spectrum — like slope sharpness or signal-to-noise — we can use that data to train a model that learns what distinguishes a successful configuration from a poor one. That model can then make informed predictions about new, unseen configurations, and even steer future optimization efforts. Because this process is grounded in data rather than manual heuristics, it avoids the pitfall of overfitting[1] to a specific experiment or run condition. Even if the simulator isn't perfectly accurate in absolute terms, the model can still learn useful patterns as long as the internal logic of the simulation remains consistent.[2]

[1] Overfitting occurs when a model learns the training data too closely, including noise or outliers, which reduces its ability to generalize to new, unseen data.

[2] If the simulator consistently applies the same rules across all inputs, the model can learn relative patterns and trends, even if absolute outputs differ from real-world measurements.

More importantly, this approach allows us not just to identify promising configurations, but to understand why they perform well. Interpretability is a core requirement — the goal is to uncover which parameters matter most, how they interact, and where the boundaries between successful and unsuccessful configurations lie. These insights feed directly into experimental planning and refinement. Later chapters will explore both the theoretical foundations and practical implementation of the machine learning models used in this framework, showing how they enable a structured, data-driven approach to optimizing dissociation experiments.

1.3 Overview of Physics Theory in the Lab

Before we can build a machine learning pipeline to classify and optimize outcomes, we need a clear understanding of what the physical system is doing — and why the structure of the experiment lends itself to simulation and data-driven modeling in the first place. This section walks through the experimental sequence in physical terms: beginning with the trap architecture and proceeding through ion loading, cooling, spectroscopy, and detection. Each stage introduces new layers of complexity and control, but also new opportunities for parameter tuning — which is where the optimization problem takes shape.

It is important to note that this section is intended primarily to offer a high-level overview of the most essential theoretical concepts underlying the lab system, and to link them together in a way that frames the simulation work presented in later chapters. It does not aim to be an exhaustive walkthrough of every physical mechanism or interaction involved. In some cases, equations or derivations are intentionally omitted or deferred to references in favor of a more focused conceptual summary. For readers seeking a more complete and detailed treatment of the system's physics and hardware implementation, we refer to Section 1.2 of Michael Mitchell's thesis [3] and the introductory chapters of Will Henshon's

thesis [5], among other works.[3] We now turn to the trap itself, beginning with the principles and structure of the linear Paul trap that forms the core of the experimental apparatus.

1.3.1 Experimental Setting: the Linear Paul Trap

The core experimental apparatus is a linear Paul trap, a well-established system for confining ions using dynamic electric fields. It consists of four long electrodes arranged in a quadrupole configuration to provide radial confinement, and two of the electrodes are segmented along trap axis to create axial confinement [6]. A high-frequency radiofrequency (RF) voltage — typically 11.04 MHz at 60 Vpp[4] — is applied to the rod electrodes, generating a rapidly oscillating quadrupolar field [3, 6].

Although this RF field is inherently time-dependent, it can be approximated as a pseudopotential in the time-averaged regime [3]. This approximation holds when the RF stability parameter q (from the Mathieu equation) is small, which is the case here [3]. The parameter q is defined as:

$$q = \frac{2e(2V_0)}{mr_0^2\Omega^2}, \tag{1.1}$$

where e is the charge of the ion, V_0 is the zero-to-peak RF voltage, m is the ion mass, r_0 is the radial distance from the trap center to the rod electrodes, and Ω is the RF drive angular frequency [4, 6]. For $^{16}O_2{}^+$, this produces a Mathieu parameter of approximately 0.05 under typical operating conditions [4].

Under these conditions, ions experience a smooth, effective harmonic potential and undergo secular motion — slow, large-amplitude oscillations in space [3]. The time-averaged pseudopotential experienced by an ion in the radial direction can be written as:

[3] Additional technical context and historical background can be found in earlier honors theses and relevant publications cited in the Bibliography.

[4] Vpp (peak-to-peak voltage) refers to the full voltage swing from the minimum to the maximum of an AC signal. For example, 60 Vpp means the voltage oscillates between −30 V and +30 V.

$$\Phi_{\text{pseudo}}(r) = \frac{e^2 V_0^2}{m\Omega^2 r_0^4} r^2 \tag{1.2}$$

where e is the ion charge, m is the ion mass, V_0 is the zero-to-peak amplitude of the applied RF voltage, Ω is the angular RF drive frequency, r_0 is the characteristic radial distance from the trap center to the electrodes, and r is the radial displacement from the trap axis.[5] This quadratic form confirms that the ion is effectively confined in a harmonic potential well, justifying the use of the secular approximation [3].

The corresponding radial secular frequency in the pseudopotential approximation is given by:

$$\omega_r = \frac{q\Omega}{2\sqrt{2}}, \tag{1.3}$$

and characterizes the strength of radial confinement for a given set of trap parameters [4].

Superimposed on this secular motion is a higher-frequency component called micromotion, which arises from the rapid oscillations of the RF trapping field [3]. Micromotion is a driven motion at the RF drive frequency Ω, and it is intrinsic to the trap geometry — any ion displaced from the RF null will experience it. In ideal conditions, where the trap is well-aligned and operated at low Mathieu q parameters (typically $q_{x,y} < 0.3$), micromotion remains small compared to the secular motion and does not significantly distort ion trajectories [3]. However, it is not captured in the pseudopotential approximation, which filters out all motion at the RF frequency.[6] Minimizing excess micromotion is critical to ensuring consistent experimental behavior and maintaining a cold, well-ordered Coulomb crystal [3].

Axial confinement is provided by static DC voltages on the endcap electrodes, creating a harmonic potential well along the trap axis [6]. The axial frequency is given by:

[5] See annotated configuration file (Appendix C) for specifics.

[6] The pseudopotential approximation assumes that micromotion is negligible and retains only the slower secular dynamics. This assumption holds near the trap center for sufficiently small q and a parameters, where secular motion dominates. A more thorough treatment of this approximation is provided in Chapter 2 of Michael Mitchell's thesis [3]

$$\omega_z = \sqrt{\frac{2\kappa eU}{mz_0^2}}, \tag{1.4}$$

where U is the applied endcap voltage, z_0 is the characteristic axial length scale, and κ is a geometric factor that depends on the trap's design [4, 6].[7] In our setup, this yields an axial secular frequency of approximately $2\pi \times 140$ kHz for O_2^+ [3, 4].

The axial confinement modifies the radial potential slightly. The corrected radial secular frequency in the presence of a non-zero ω_z is:

$$\omega_{x,y} = \sqrt{\omega_r^2 - \frac{\omega_z^2}{2}}. \tag{1.5}$$

as defined in [4]. Together, the RF and DC fields define three secular frequencies — one in each spatial direction — which depend on the ion's charge-to-mass ratio q/m. This species-dependent variation in motional frequencies is critical for selective excitation, enabling the experiment to target specific ion types through resonant modulation [3].

The overall trap geometry is compact — typically a few millimeters in each dimension — which ensures tight spatial confinement and strong inter-ion Coulomb interactions [3, 5, 6]. These conditions are essential for efficient cooling, stable Coulomb crystal formation, and the indirect detection techniques employed later in the experiment. With the trap architecture in place, we now consider how the system is populated with ions and how those ions evolve into a stable crystalline configuration suitable for spectroscopy and detection.

1.3.2 Ion Loading and Crystal Formation

Once the trap fields are established, the system is loaded with three ion species: Be^+, O_2^+, and O^+. Be^+ is introduced by resistively heating a beryllium oven — a thin wire wrapped around a tungsten core — to sublimate atomic beryllium, which is then photoionized using a 235 nm laser beam derived from a frequency-doubled ECDL [3, 7]. Molecular oxygen (O_2)

[7] As before, these values can be found in the Annotated Configuration file in Appendix C.

is introduced as a neutral gas pulse, directed through a series of skimmers into the trap region, and ionized via a 301 nm pulsed laser using a 2+1 REMPI process [2, 3, 5]. O^+ ions are not injected directly; instead, they appear as a product of O_2^+ dissociation during later experimental stages.[8] The resulting multi-species Coulomb crystal is stably confined and sympathetically cooled via Doppler laser cooling of Be^+, enabling precision measurements via fluorescence readout.

After ionization, the resulting plasma is confined by the trap and begins to evolve under the influence of the pseudopotential and inter-ion Coulomb repulsion [3]. As energy is removed (via processes discussed in the next subsection), the ions settle into an ordered, low-temperature structure known as a Coulomb crystal [3]. This is not a crystal in the traditional solid-state sense, but rather a spatially stable arrangement of ions in which their mutual repulsion balances against the confining potential [3, 5]. The repulsive force between ions is governed by the Coulomb potential:

$$U_{ij} = \frac{q_i q_j}{4\pi\varepsilon_0 |\mathbf{r}_i - \mathbf{r}_j|}, \tag{1.6}$$

where q_i and q_j are the charges of ions i and j, and $|\mathbf{r}_i - \mathbf{r}_j|$ is their separation. Minimizing this interaction energy — while confined in the external trap potential — leads to the formation of the crystal structure [3].

The crystal's spatial structure is governed by the q/m ratio of each species. Ions with a higher charge-to-mass ratio — such as Be^+ — experience stronger confinement and settle closer to the center of the trap [3, 4, 6].[9] This radial position r is determined by a balance between the trap pseudopotential and the Coulomb repulsion. For an ion of mass m and charge q, the equilibrium position approximately satisfies:

[8] Each dissociation event converts one O_2^+ ion into a single O^+ product ion, making the presence and number of O^+ ions a direct readout of dissociation yield.

[9] This spatial ordering allows for indirect sensing: because Be^+ resides at the center and fluoresces, its motion reflects perturbations caused by outer ions like O_2^+ and O^+, enabling indirect detection through Coulomb coupling.

$$\frac{q^2 V_0^2}{m\Omega^2 r_0^4}\,\mathbf{r}_i \sim \sum_{j\neq i} \frac{q^2}{4\pi\varepsilon_0 |\mathbf{r}_i - \mathbf{r}_j|^2}, \tag{1.7}$$

demonstrating that lighter, more strongly confined ions remain near the center, while heavier species like O_2^+ and O^+ are pushed to the edges [3]. This radial separation is critical for later stages of the experiment, as it enables motion in one species to be indirectly sensed through another via Coulomb coupling. To maintain this delicate spatial arrangement and ensure long-term stability of the crystal, active cooling mechanisms must be applied to dissipate residual thermal energy and suppress unwanted motion.

1.3.3 Laser Cooling and Doppler Dynamics

To stabilize the ion crystal and reduce its thermal energy, we apply Doppler cooling to the Be^+ ions [3, 4]. A near-resonant laser is directed at the crystal with a detuning slightly below the Be^+ resonance frequency [2, 7, 8]. Ions moving toward the laser experience a Doppler shift that brings them closer to resonance, increasing the scattering rate [3, 7, 8]. This results in a net damping force that opposes the ions' motion, cooling them toward the Doppler temperature limit. The photon scattering rate is given by:

$$R_{\text{scat}} = \frac{s_0 \Gamma/2}{1 + s_0 + \left(\frac{2\delta}{\Gamma}\right)^2}, \tag{1.8}$$

where s_0 is the saturation parameter, Γ is the natural linewidth of the transition, and δ is the effective detuning, including the Doppler shift due to ion velocity [9].

The laser is defined by several parameters: beam origin, direction, waist size, detuning, and saturation intensity. These values are carefully chosen to maximize cooling efficiency along all spatial axes, with emphasis on the axial direction where secular motion is most strongly observed [3].

Only Be^+ interacts with the laser directly. O_2^+ and O^+ are optically dark, meaning they do not scatter photons and therefore cannot be cooled by the laser itself [4]. Instead, they

undergo sympathetic cooling via Coulomb interaction with Be^+ [3, 10]. As long as the crystal is well-coupled, thermal energy is redistributed across all species, bringing the entire system to a stable low-temperature state [10].

This cooling phase usually lasts a few milliseconds and is essential for reaching the crystalline regime [3]. Without it, the ions remain in a high-temperature plasma state and do not exhibit the spatial structure or motional stability needed for precise dissociation detection.[10] Once the ions are sufficiently cooled and the crystal is stabilized, the system is ready for controlled dissociation events, which form the basis of the signal we aim to detect and optimize.

1.3.4 Dissociation of Molecular Oxygen

In our experiment, molecular oxygen ions (O_2^+) are dissociated using either a 266 nm or 355 nm laser after vibrational excitation [2]. Each dissociation event yields a single trapped O^+ ion. The number of O^+ ions observed in the trap serves as the experimental readout for dissociation yield. We do not simulate the dissociation dynamics directly; instead, QLICS initializes the system with a fixed number of O_2^+ and O^+ ions, allowing us to focus on optimizing downstream detection [11].[11] To detect these dissociation events with high fidelity, we rely on a modulation technique that selectively excites specific ion species and reveals their presence through induced motion within the crystal.

1.3.5 Modulation and Fluorescence Mass Spectrometry

After the system has cooled into a stable crystal, we apply a modulation field — a time-dependent electric field — to drive species-specific excitation. This is implemented by mod-

[10] Precise dissociation detection relies on observing subtle fluorescence changes; excess motion from an uncooled or unstable crystal would mask these signatures, reducing resolution and obscuring stepwise ion loss.

[11] Because the dissociation process is not the primary focus of this study, we bypass simulating the molecular dynamics of bond breaking with QLICS. Instead, it assumes idealized dissociation outcomes to isolate and optimize the downstream detection signal — the key experimental observable.

ulating the trap potential at a user-defined frequency, amplitude, and spatial profile. The field is typically represented using a second-order polynomial expansion, with coefficients precomputed from SIMION[12] electrostatic simulations [3, 12].

The key idea is to scan the modulation frequency across a range that includes the secular frequencies of the target species. When the modulation matches the resonant frequency of a particular ion type (e.g., O_2^+ or O^+), it drives coherent oscillations in that species [12]. The applied electric field generally takes the form:

$$E_x(t) = (A_0 + A_1 x + A_2 x^2) \cos(2\pi f_{\text{mod}} t), \tag{1.9}$$

where A_0, A_1, and A_2 are field coefficients, and f_{mod} is the modulation frequency [3]. This form allows for controlled spatial and temporal targeting of the excitation field [3].

These motions, while not directly observable, perturb the Be^+ ions via Coulomb coupling. This is the essence of LCF-MS spectroscopy: by applying a weak, frequency-swept modulation and monitoring the Be^+ response, we can infer motion in another species. The modulation must be carefully tuned — strong enough to induce motion, but not so strong that it disrupts the crystal or causes ion loss [3]. The waveform duration, amplitude, and shape are all tunable parameters in the experiment.

Because Be^+ is the only species that interacts with the detection laser, the entire detection mechanism relies on this indirect motional coupling [4]. The next phase reads out the resulting motion using Be^+ fluorescence.

1.3.6 Fluorescence Detection and Readout

Following modulation, a near-resonant laser is used to induce fluorescence in the Be^+ ions [3, 12]. A photomultiplier tube (PMT) collects photons scattered along a defined solid angle, and the photon counts are recorded over time [3]. This produces a PMT trace — a time-

[12] SIMION is a charged-particle optics simulation program used to calculate electric field distributions from electrode geometries. In this context, it provides the spatial field coefficients used to construct the modulation waveform applied in the experiment.

series dataset representing the instantaneous scattering rate of Be^+ ions during the detection window [3].

The detection mechanism exploits Doppler dynamics: motion of the Be^+ ions shifts their absorption profile relative to the cooling laser, suppressing photon scattering. When O_2^+ ions are resonantly excited, their motion perturbs the Be^+ ions, increasing their velocity and reducing fluorescence. This results in a dip in the photon count rate — a signature that resonant motion has occurred in the target species [3, 4, 12].

A high-quality dissociation signal is characterized by a sharp, monotonic drop in fluorescence at a specific modulation frequency, often accompanied by a clear slope in a frequency sweep. In contrast, poor signals appear flat or noisy, indicating weak coupling or off-resonant excitation. The steepness and clarity of the fluorescence trace define our experimental figure of merit, and form the basis for the cost functions and classifiers developed later in this thesis.

Critically, this detection method is fully indirect. We never observe O_2^+ or O^+ ions directly — all measurements are inferred from their dynamical effects on Be^+ [3]. This makes the system highly sensitive to experimental parameters like alignment, cooling quality, and trap stability. Even small deviations in modulation amplitude or endcap voltage can significantly degrade signal quality.

1.3.7 Theory Summary

To summarize, the experimental platform centers around a linear Paul trap, where radial confinement is provided by oscillating RF voltages and axial confinement by static DC voltages. These fields collectively create a pseudopotential that confines ions according to their charge-to-mass ratio, producing species-specific motional frequencies. Once loaded, ions settle into a Coulomb crystal structure determined by both the trap geometry and inter-ion repulsion, with lighter species occupying the center and heavier ions pushed outward. Laser cooling is applied to Be^+ ions to reduce thermal motion, enabling stable crystal formation.

Other species, including O_2^+ and its dissociation product O^+, are cooled sympathetically through Coulomb interactions.

Dissociation is triggered optically and results in additional O^+ ions, which alter the crystal's motional dynamics. These changes are indirectly detected by applying a species-targeted modulation field and observing the fluorescence response of the Be^+ ions. A decrease in fluorescence indicates resonant excitation, and the slope of the resulting photon count trace serves as a proxy for dissociation resolution. Because the system's readout is indirect and depends on careful alignment and parameter tuning, it is highly sensitive to fluctuations. It is precisely this sensitivity that motivates the need for simulation.

1.4 Why Simulation?

This is where simulation becomes indispensable. In principle, one could collect all necessary data from physical experiments, manually adjusting parameters and recording outcomes. In practice, however, lab experiments are time-consuming, resource-intensive, and prone to disruption. Small misalignments, ambient temperature fluctuations, or beam instabilities can compromise repeatability and introduce noise [3].[13]

To circumvent these limitations, we turn to QLICS[14] — the Quantum Logic Ion Control Simulator — a custom wrapper around LAMMPS that produces high-fidelity simulations of the experimental sequence. Given an `.ini` configuration file specifying the ions, trap geometry, laser properties, detection setup, and modulation phases, QLICS simulates time evolution and returns photon count traces that closely resemble those generated by the photomultiplier tube (PMT) in the actual experiment [3, 11]. These synthetic traces form the dataset used to train and evaluate all subsequent models.

Importantly, the goal is not to make QLICS a flawless replica of the real system. Rather,

[13]Experimental signals can be disrupted by small fluctuations in alignment, ambient conditions, or beam path integrity. As noted in Michael Mitchell's thesis, issues such as missing resonances, unstable peaks, and unexplained signal behavior motivated the development of QLICS to provide controlled, repeatable simulations that bypass these sources of experimental instability [3].

[14]Thank you to Michael Mitchell for development and testing of QLICS.

we require that it be internally consistent — meaning that for any given set of input parameters, it produces output signals that are structured, repeatable, and physically sensible. Perfect fidelity to the real experiment is neither expected nor necessary; what matters is that the simulator reliably captures the dominant dynamics and trends relevant to experimental outcomes.[15] The machine learning model is not tasked with uncovering the fundamental physics from first principles;[16] it only needs to learn the behavioral structure encoded by QLICS. In this sense, QLICS functions as a fast, controllable sandbox for exploring parameter space and prototyping optimization strategies before committing to time-intensive experimental runs.

This framework sets the stage for a broader motivation. While QLICS provides a simulated environment for optimization, the relevance of this effort hinges on the foundational role that ion traps play in modern experimental physics. Understanding the general importance of trapped ion systems helps contextualize why efficient control and optimization of these platforms matters at all — and why simulation-based tools like QLICS are increasingly indispensable.

1.5 General Importance of Ion Trapping

Trapped ion systems have emerged as one of the most powerful and tunable platforms in modern experimental physics [10, 13]. They serve critical roles in the field of quantum information, ultrahigh-resolution spectroscopy, and precision tests of fundamental constants [12, 14]. Their appeal lies in the fine-grained control they afford: over individual particles, over the geometry and strength of electromagnetic fields, and over dynamical processes across multiple time and energy scales [10, 14]. That same tunability, however, introduces a significant degree of complexity. As the number of adjustable parameters grows, so does the difficulty

[15] In this setting, internal consistency ensures that trends learned by the model reflect true parameter dependencies rather than stochastic noise, making the simulator a dependable training environment.

[16] In other words, the model learns the simulator's input-output mapping, not the complete mechanistic details of ion motion or quantum state transitions.

of identifying configurations that yield high-quality experimental outcomes [3]. This thesis addresses that challenge directly — not simply as a machine learning study or a simulation exercise, but as a blueprint for turning complex experimental design into a well-posed, algorithmically tractable optimization problem.

Beyond its methodological contributions, this work supports a broader scientific goal: testing the long-term constancy of the proton-to-electron mass ratio, μ, via high-resolution spectroscopy of vibrational transitions in O_2^+ ions. While the Standard Model predicts μ to be invariant, certain extensions — including string-theoretic and scalar-field models — allow for slow cosmological drift [1]. Observing even a minute deviation would constitute direct evidence of new physics.

To enable such tests, the experiment must detect dissociation events with exceptional precision. That means resolving the number of O^+ ions produced in each trial cleanly and reproducibly — a task that hinges on careful control of trap parameters, modulation conditions, and fluorescence readout. Optimizing these elements systematically is what transforms the platform from a basic spectroscopy setup into a viable probe of fundamental constants.

Subsequent chapters will examine how classification models are trained, how performance is validated, and how those models integrate with an optimization pipeline that actively proposes new configurations. But the underlying motivation remains consistent throughout: to increase resolution in molecular dissociation measurements, to reduce experimental trial-and-error, and to build a self-correcting system that improves over time. Achieving those goals is not merely a technical contribution — it supports the broader objective of testing fundamental symmetries and potentially uncovering time variation in μ, which would mark a profound departure from currently accepted physical law. To reach that level of experimental precision, however, we must first understand the simulation framework that underlies prediction and classification in this thesis.

Chapter 2

Simulation Architecture

Before diving into machine learning optimization, it's necessary to unpack the mechanics of the simulation itself — what's actually being simulated, how it's structured, and what the simulation environment does when we say "run an experiment." Simulation is the backbone of this project. Without a clear picture of how our simulations work under the hood, the later machine learning pipeline would just feel like tuning knobs on a black box. The goal here is to make that box transparent. To that end, the next section introduces QLICS — the software framework that underpins all simulations in this study. By walking through its structure and execution flow, we establish the foundation needed to understand how experimental behavior is modeled, analyzed, and ultimately used for training downstream machine learning algorithms.

2.1 QLICS Overview

QLICS is a standalone Python application designed to simulate a trapped ion experiment based on a user-specified configuration [3]. Each simulation begins with a human-readable `.ini` file that defines experimental parameters, such as trap voltages, ion species, modulation waveforms, laser geometries, and detection settings. The file structure is parsed into Python dictionaries and interpreted by QLICS into a sequence of commands that prepare, run, and

analyze a virtual experimental trial [3, 15].

QLICS serves as an orchestration layer. It handles file parsing, phase scheduling, object initialization, parameter iteration, and output formatting [15]. Crucially, it also wraps around the LAMMPS engine, injecting domain-specific physics (e.g., laser cooling, pseudopotential approximations) and coordinating LAMMPS's time evolution with experimental phases [3, 11]. The architecture is modular.[1] Each `.ini` file defines:

→ Ion clouds with species, counts, and initial radii.

→ Trap configurations using geometric and RF parameters.

→ Cooling and detection lasers with full optical profiles.

→ Modulation fields expressed as low-order electric field expansions.

→ Timing sequences specifying when each force or signal is active.

Together, these allow the simulation of experiments with time-resolved control over the physics applied to the system. To implement the core physical dynamics of ion motion and interaction, QLICS relies on an external simulation engine capable of high-precision numerical integration.

2.2 LAMMPS Integration

QLICS utilizes LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator)[2] as the physics engine responsible for computing inter-ion forces and integrating the classical equations of motion [11]. Originally developed for simulating atomic and molecular systems, LAMMPS is both efficient and extensible, making it a suitable backend for trapped-ion simulations — provided we supply the correct physical approximations [11].

[1] Modular here means that each component of QLICS — including ion initialization, cooling routines, modulation application, and readout — is implemented as a discrete, interchangeable module. This allows users to modify, extend, or replace individual parts of the simulation pipeline without changing the entire framework.

[2] For specific documentation, see [11]

LAMMPS is used for: (i) Calculating inter-ion Coulomb forces, (ii) evolving the system via the velocity-Verlet integrator [3, 16],[3] and (iii) applying externally defined forces such as cooling, modulation, and trap confinement. However, LAMMPS has no native understanding of trapped-ion physics. To adapt it, QLICS injects several layers of abstraction [3]:

→ Trap forces are implemented using an effective harmonic potential (i.e., the pseudopotential approximation).

→ Laser cooling forces are applied via custom fix objects that calculate Doppler shifts and velocity-dependent damping.

→ Electric field modulations are computed analytically from polynomial field expansions and applied as time-dependent external forces.

LAMMPS remains unaware of the experimental semantics (e.g., modulation phase or detection time).[4] QLICS controls which forces are active at each phase and passes updates to LAMMPS accordingly. This makes LAMMPS an efficient but agnostic worker engine — all context resides in QLICS.

2.3 Simulation Workflow

Each simulation consists of a defined sequence of time phases, each with its own timestep, duration, and active physical processes. These are listed in the [`timesequence`] block of the `.ini` file and executed in order [3]. The typical life cycle involves:

1. **Initialization**

   Ion positions are sampled randomly within user-defined cloud radii. These form the starting conditions of the simulation. Velocities are initially set to zero [3].

[3] Formulation is noted in detail in the appendix of Michael Mitchell's thesis [3].

[4] LAMMPS acts like the engine of a car: it executes the physics computations efficiently, but it doesn't know where it's going or why. QLICS is the driver — responsible for steering, scheduling, and interpreting the experimental context.

2. **Trap Thermalization Phase**

   Cooling lasers are applied to thermalize the ion crystal. LAMMPS evolves the system under trap and laser forces. This phase ensures ions settle into a realistic configuration before modulation or detection.

3. **Modulation Phase**

   Time-varying electric fields defined in `[modulation_*]` blocks are applied. These fields are calculated from polynomial expansions (e.g., `ex0`, `exx1`, `exx2`) derived from SIMION field maps [17]. The fields oscillate at a user-defined frequency, designed to resonantly excite motion in target species.

4. **Detection Phase**

   A near-resonant scattering laser is applied to Be^+ ions. The Doppler-shifted scattering rate is calculated for each ion, timestep-by-timestep. Photons emitted along a specified direction are collected into a synthetic PMT signal — a time-series CSV output [3].

5. **Post-Processing**

   After each run, QLICS analyzes the resulting trace. If an `[iter]` block is defined (e.g., for frequency sweeps), QLICS loops over parameter combinations, modifies the .ini inputs accordingly, and generates a batch of simulations [3, 15].

2.4 Stochasticity

While QLICS is designed to be deterministic in its force modeling and numerical integration, several key aspects of each simulation are intentionally stochastic, in order to better reflect the probabilistic nature of real experimental systems.

First, ion initialization introduces randomness at the very start of each simulation. When an ion cloud is defined in the `.ini` file, its constituent ions are randomly positioned within a specified spherical radius around the trap center [3, 15]. Though the number of ions and initial cloud size are fixed, the exact spatial distribution of ions varies between runs,

producing slightly different initial Coulomb interactions and thermalization paths. This affects not only crystal structure but also energy distribution[5] at the end of the cooling phase [11].

Second, the direction of each emitted photon is also randomized using isotropic angular sampling (or with a detector-specific angular cutoff if defined), and only photons within the collection solid angle are recorded [3]. These stochastic draws introduce run-to-run variability in the final PMT trace — even when the same `.ini` file is used.[6]

Third, thermalization dynamics are indirectly stochastic due to their dependence on initial conditions [11]. Although laser cooling is modeled deterministically via Doppler damping, the random ion positions and subsequent Coulomb collisions during early time evolution cause variability in how quickly and cleanly the crystal forms, especially in configurations close to melting thresholds [11, 15]. This impacts the effective temperature and spatial coherence of the crystal when modulation begins.

Finally, modulation responses can be sensitive to these stochastic elements [3, 4, 12]. Because the system is driven near resonances, slight differences in initial crystal alignment or thermal state can amplify into larger differences in ion trajectories during the modulation phase. This is especially true when a secular frequency is only marginally matched, or when motional coupling between species is highly state-dependent [3].

Together, these stochastic components create a simulation environment that is not entirely deterministic — and intentionally so. Rather than producing identical outputs for fixed inputs, QLICS is designed to simulate the statistical distribution of possible outcomes that one might observe in a real experimental run. This variability is not a source of error but a feature, and it informs how downstream machine learning models must be trained: not just to predict point outcomes, but to be robust to uncertainty, noise, and run-to-run variation. Later chapters will leverage Monte Carlo sampling to quantify this variability and

[5]Small differences in initial ion spacing lead to variation in how energy is redistributed during Doppler cooling. This affects final ion velocities, local heating, and the overall symmetry of the resulting crystal structure — all of which can subtly alter modulation response and fluorescence output.

[6]This is why simulated photon counts are often non-integer values [3].

to build models that account for it explicitly.

2.5 File Format and Parameter Definitions

All simulation inputs are defined in a single `.ini` file, structured by sections [15]:

- → `[ions]` – Defines species, charges, masses, and cooling behavior.
- → `[ion_cloud_*]` – Specifies ion counts and initial conditions.
- → `[trap_*]` – Sets trap geometry, RF frequency, and voltages.
- → `[cooling_laser_*]` – Defines laser position, beam waist, detuning, and k-vector.
- → `[modulation_*]` – Configures modulation fields and their time dependence.
- → `[detection]` – Specifies photon collection angles and efficiencies.
- → `[timesequence]` – Declares time phases, steps, and active processes.
- → `[iter]` – Encodes sweeps over experimental parameters (e.g., tickle frequency).

These sections are parsed by QLICS into Python dictionaries, validated, and used to build the LAMMPS input objects [11].

2.6 Use of Pseudopotential Approximation

One key simplification is that QLICS uses the pseudopotential approximation to represent the time-averaged effect of the RF trap [3]. Rather than simulating RF micromotion directly (which would require 10–100x smaller timesteps),[7] we treat the trap potential as a static harmonic well defined by:

$$\Phi(x, y, z) = \frac{1}{2} m \left(\omega_{x,y}^2 (x^2 + y^2) + \omega_z^2 z^2 \right) \tag{2.1}$$

where ω_r and ω_z are computed from the user-defined RF and DC voltages, as well as the ion's charge-to-mass ratio [3].

[7] As described in [3].

This allows QLICS to evolve systems on the scale of nanoseconds to milliseconds with manageable computational cost, while preserving the essential physics that determines secular frequencies and species separation [3]. With time evolution handled by LAMMPS and parameter logic managed by QLICS, the final step is to extract observable signals that mirror what an experimenter would detect in the lab.

2.7 Output Format and Trace Generation

The final output of each simulation is a CSV file mimicking the signal from a PMT detector. The data include: (i) timestamps (in seconds), (ii) photon counts per timestep, and (iii) metadata (e.g., iteration index, frequency, ion configuration) [15].

As a reminder from the introductory chapter, photon counts are calculated based on each Be^+ ion's velocity and laser alignment, using:

$$R_{\text{scat}} = \frac{s_0 \Gamma}{2\left(1 + s_0 + \left(\frac{2\delta}{\Gamma}\right)^2\right)} \tag{2.2}$$

with relativistic Doppler shifts applied to get $\delta(v)$ for each ion [9].

For iterated simulations (e.g., frequency sweeps), QLICS enables us to assemble a directory of CSVs,[8] one per run, which are later aggregated for analysis and ML model training. Once simulations are run and outputs are collected, the next challenge lies in scaling this process to explore large parameter spaces efficiently.

[8] In addition to the primary `ph_scattering.csv` file, QLICS outputs include a `positions.txt` file (logging ion positions and velocities at each timestep), LAMMPS logs (for debugging and performance tracking), and an optional `state_history.pkl` file storing simulation metadata and intermediate states. These outputs collectively enable post-simulation diagnostics, trajectory analysis, and full trace reconstruction [15].

2.8 HPC and Parallelization

2.8.1 Design Philosophy

The architecture of QLICS is intentionally designed to support this kind of parallelism. Each simulation: (i) reads only from its own .ini file, (ii) writes to its own output path, (iii) and carries no memory of global state or other runs. This statelessness makes it trivial to scale without inter-process communication, and enables reliable parallel execution, whether in a batch job or across a local thread pool. The entire simulation pipeline is effectively embarrassingly parallel — a term used in high-performance computing to describe problems that are easily distributed without complex interdependencies.[9]

2.8.2 Execution Strategy and Modifications

To take advantage of QLICS's parallel-friendly design, two modes of execution are used throughout this thesis: local `multiprocessing` and HPC-based job arrays, where HPC denotes High-Performance Computing.[10]

For batch processing in particular, a modified version of QLICS — called `qlics_batch` — was developed in place of the standard interactive version. Unlike the main QLICS program, which presents the user with a menu for selecting single-run configurations, `qlics_batch` is hardcoded to execute batch runs. It bypasses all interactive input and includes additional infrastructure to optimize the running of many `.ini` files concurrently. This makes it especially well-suited for high-throughput operation on HPC systems using `Slurm`.[11]

Instead of relying on typical batch runs in QLICS which sequentially run files one after

[9] Because each QLICS run is fully independent — with no shared memory, synchronization, or cross-process data exchange — there are no complex interdependencies between simulations. This allows them to be executed concurrently on separate threads or nodes without coordination overhead, fulfilling the definition of an embarrassingly parallel task [15].

[10] Parallelization refers to the execution of many simulations at once, either across CPU cores on a single machine (via `multiprocessing`) or across compute nodes in a cluster (via HPC job arrays) [18].

[11] `Slurm` (Simple Linux Utility for Resource Management) is a workload manager used to schedule and distribute jobs across HPC compute nodes [18].

another, we utilize Python's built-in `multiprocessing` module, which governs parallelization across CPU cores irrespective of HPC use.[12] This approach works well for small sweeps or local development, given that typical laptops provide enough compute power to parallelize 16 or so simulations at once.

For large-scale sweeps or Monte Carlo trials, simulations are executed on Amherst's HPC cluster using `qlics_batch` and launched as Slurm job arrays [18]. Each array index corresponds to a distinct simulation defined by a separate configuration file, and outputs are directed to a scratch directory for efficient I/O.[13] This setup supports thousands of simulations running in parallel and enables rapid[14] dataset generation [18].

Both approaches share the same backend logic in QLICS, and switching between them is seamless. The key requirement is that each simulation run be stateless and independent — a condition QLICS enforces by design. With both local and high-performance computing strategies in place, we are able to generate large, structured datasets of dissociation outcomes under a wide range of parameter configurations. These datasets form the backbone of our learning framework — providing the input-output pairs required to train models that can generalize experimental behavior and guide future optimization. Equipped with this foundation, we now shift to the theoretical underpinnings of machine learning and the rationale behind its application to this system.

[12] Output paths are assigned dynamically to prevent file collisions.

[13] I/O stands for input/output — the process by which data is read from or written to files, memory, or storage devices. Efficient I/O management is crucial in high-performance computing to minimize bottlenecks during large-scale simulations.

[14] By rapid, we mean that without high-throughput parallelization on HPC resources, generating this volume of data would take several years on a standard single-core machine.

Chapter 3

Machine Learning Background and Theory

Machine learning, broadly speaking, is a method of using past data to make future predictions. It's an umbrella term that encompasses a wide variety of algorithmic approaches, but at its core, the field focuses on one fundamental idea: how can we use previously observed input-output pairs to learn an underlying pattern or function, and then use that learned function to predict or classify new data we haven't seen before? (In this context, a "model" refers to a mathematical or computational structure — often a set of equations, trees, or functions — that attempts to approximate the relationship between inputs and outputs [19]. "Training" a model simply means adjusting its internal structure using a dataset, so that it performs well at capturing those relationships [19]. Once trained, the model can generalize its learned behavior to new inputs it hasn't seen before.) The value in this approach is especially evident in complex, high-dimensional systems, where traditional analytical modeling might fail due to noise, incomplete information, or sheer parameter space overload [19, 20]. You may be wondering why we might choose to use machine learning in the context of this study to begin with. There are a multitude of reasons: First, the characteristics of complex systems are precisely the conditions that arise in trapped ion simulations — where dozens

of control parameters (cooling times, laser amplitudes, trap voltages, tickle timing, ramp sequences, and so on) interact in non-trivial ways, producing outputs that are difficult to predict or optimize through direct intuition alone. In addition, the simulation output itself is often noisy or nonlinear, making traditional gradient-based methods either unreliable or entirely inapplicable. Machine learning models, especially those tailored for classification, can operate effectively even when the cost landscape is jagged, discontinuous, or sparsely sampled — which is exactly the regime we're working in. On top of that, once trained, a machine learning model can generate near-instant predictions for new configurations, allowing for rapid, iterative exploration of the parameter space without having to rerun full simulations each time. This kind of real-time feedback is essential when dealing with optimization problems where simulation runtime is non-negligible and experimental repeatability is far from guaranteed.

The power of machine learning lies in its ability to act as a dynamic learner — not just a calculator, but a system that can pick up patterns, generalize from examples, and improve its predictions as more data becomes available [20, 21]. In a way, it's exactly like onboarding a new lab partner who doesn't need a full theoretical derivation every time — just enough labeled examples to figure out what "good" looks like. But unlike a human, this partner can sift through thousands of simulations, recognize subtle correlations across high-dimensional parameter sets, and respond instantly when asked whether a new configuration is likely to work [19, 21]. This makes machine learning especially valuable in our context, where outcomes depend on complex parameter interactions and full simulations are expensive[1] to compute. Rather than treating each run in isolation, the model builds a memory — a structured internal map of what works and what doesn't — and uses that to guide further exploration [19].

In our specific context, machine learning provides a data-driven way to automate and guide the search for optimal configurations. Rather than performing manual tuning of input

[1] In this context, "expensive" refers to the computational cost of running full QLICS simulations — each one taking several minutes at minimum — which becomes prohibitive when exploring large parameter spaces.

parameters based on trial and error, we can systematically collect simulation data (i.e., outcomes for different parameter configurations), then use that data to train a model that "learns" what makes an experiment good or bad. Of course, "good" is not some abstract or aesthetic label — we define it rigorously in a future chapter based on a cost function that reflects the quality of number resolution, particularly slope steepness and consistency in dissociation graphs. So at a high level, the machine learning system we're constructing is designed to take in a new set of parameter values and output a classification: is this a "good" experiment, or not?

Before jumping into the details of the algorithms we'll use, it's important to distinguish between the broad types of machine learning. While there are many taxonomies, the two most relevant to our case are supervised learning and unsupervised learning. Supervised learning involves labeled data — in our case, each set of parameters results in a series of outcome values which are used to calculate whether that simulation is good or not [19]. We use this labeled dataset to train a model to make similar predictions for new parameter combinations [21]. Within supervised learning, we're particularly interested in classification tasks. This means the model isn't predicting a continuous number (like a temperature or time), but instead assigning one of a finite number of categories — "good" or "bad" in our binary classification framework [20]. This is distinct from regression, which deals with continuous outputs [20]. On the other hand, unsupervised learning, like clustering, finds patterns in data without any labels — which may become relevant in future analysis, but is not the focus here [20].

Since this thesis is mainly focused on classification tasks, it is important to discuss the theory behind the forms of supervised learning utilized in the study. Let's now walk through the specific algorithms that we'll be using in the rest of this thesis, beginning with the fundamental building block of the random forest model.

3.1 Decision Trees

To truly understand random forests, it helps to start with the foundation: the decision tree. A decision tree is one of the most intuitive, interpretable, and surprisingly powerful models in the machine learning toolbox [19]. At a high level, a decision tree is exactly what it sounds like — a tree-like structure that recursively splits the dataset into smaller and smaller chunks, based on rules that aim to simplify the classification task [19]. Each node in the tree represents a decision based on a particular feature (i.e., a parameter in the simulation), and each "leaf" at the bottom of the tree assigns a final label or outcome — in this case, something like "good" or "bad" experiment [19]. The following outlines how such a tree is actually built.

Suppose the dataset consists of a multitude of simulated experiments. Each data point corresponds to a full simulation run: a specific configuration of input parameters — such as tickle frequency, endcap voltage, cooling ramp length, etc. — along with an output label indicating whether the simulation produced a useful result or not. The goal of a decision tree is to classify a new, unseen configuration as "good" or "bad," based on what it has learned from past data.[2]

The process begins at the root node — ironically, the *top* of the tree in these models as shown in Figure 3.1. At this point, the model has access to the entire dataset and must decide how to split it in a way that moves closer to making a correct classification [19]. It does this by evaluating all possible features (i.e., simulation parameters) and all possible values they could be split on, selecting the one that yields the highest "information gain [19, 20]." This is a quantitative measure of how much a split reduces the uncertainty or "messiness" of the data [19].

Formally, information gain is computed as:

[2]We will henceforth use "learn" to refer to the process of identifying patterns in the training data [19].

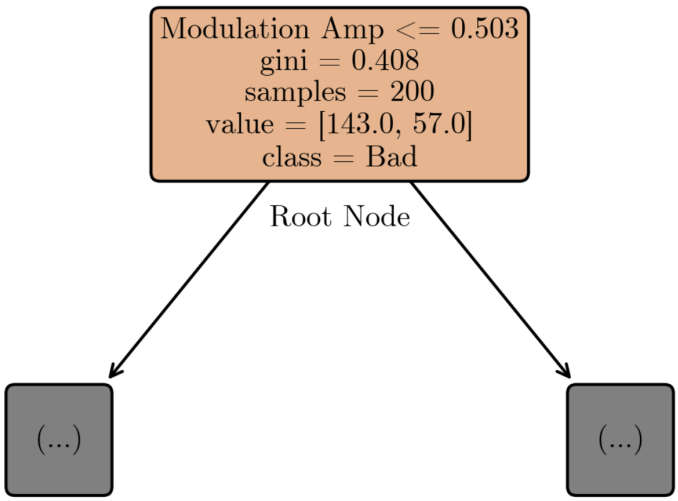


Figure 3.1: Root node of a decision tree trained on simulated experimental data. The split condition (`mod_amp` $\leq$ `0.48`) indicates the threshold used to divide the dataset. "Gini" quantifies class impurity in the node: 0 means perfectly pure, and 0.5 indicates maximum uncertainty between two classes. "Samples" refers to the number of data points reaching the node. "Value" shows the class counts (bad vs. good), and "Class" indicates the majority class assigned for prediction.

$$IG = H(S) - \sum_{i=1}^{n} \frac{|S_i|}{|S|} H(S_i) \tag{3.1}$$

where $H(S)$ is the entropy of the parent set, and each S_i is a subset resulting from the split [19]. The weighted average of the child entropies is subtracted from the parent entropy to determine how much uncertainty was reduced [19].

To measure this messiness, metrics such as Gini impurity or entropy[3] are commonly used [19, 20]. Both are functions that increase as the classes become more mixed (e.g., 50% good, 50% bad) and decrease as the data becomes more homogeneous (e.g., 95% good, 5% bad). Gini impurity, for instance, is defined as:

$$G = 1 - \sum_{i=1}^{C} p_i^2 \tag{3.2}$$

[3]Entropy weights rare class probabilities more heavily than Gini, which approximates class mixing with a simpler quadratic form [19].

where p_i is the proportion of class i in the node, and C is the total number of classes (just two in this case) [19]. A node with only one class has $G = 0$, indicating perfect purity [19].

Consider a split at a tickle frequency of 420 kHz. The dataset is divided into two groups: all simulations with tickle frequency < 420 kHz, and those ≥ 420 kHz. The model examines the composition of "good" vs. "bad" experiments in each group, computes the impurity in each, and averages them (weighted by how many points are in each group). This yields the impurity after the split. The tree compares this to the impurity before the split and selects the option with the greatest reduction — the most information gained [19, 22].

Once the best feature and split point are selected, the tree creates two new child nodes — one for each subset of the data — and recursively[4] repeats the process in each one [22, 23]. Each node now operates on a smaller, more homogeneous portion of the data. This recursive partitioning continues until a stopping condition is met. Common stopping criteria include:

- → All the examples in the node are from the same class (perfect purity).
- → The number of examples in the node falls below a minimum threshold [19, 23].
- → The maximum allowed depth of the tree is reached.
- → The gain from further splitting becomes negligible.

When a node stops splitting, it becomes a leaf, and the tree assigns it a final prediction — usually the majority class of the data points in that leaf [22]. For example, if a leaf contains 90% "good" examples and 10% "bad," it is classified as "good." The prediction can also return probabilities — in this case, a 90% confidence in a good outcome [23].

Fundamentally, the decision tree is slicing up the parameter space — the multidimensional space defined by all simulation inputs — into rectangular regions, each corresponding to a different sequence of decisions [19]. For example, one region might correspond to: tickle frequency $\geq 420\,\text{kHz}$, endcap voltage $< 4.2\,\text{V}$, cooling ramp time $\geq 0.5\,\text{ms}$, and so on. Any new simulation with inputs falling into that region receives the same classification. These

[4]Recursively here means the tree-building process is applied repeatedly to each new subset, with the same logic used at every level of the tree structure.

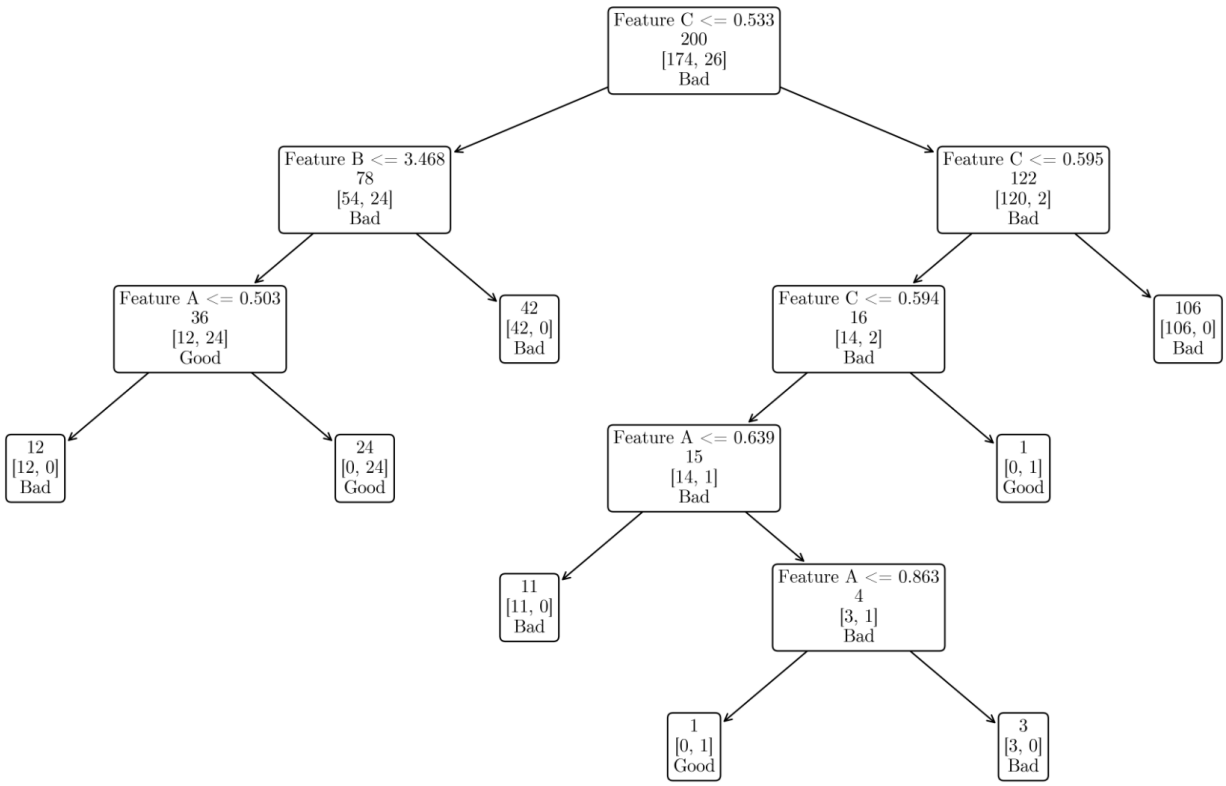


Figure 3.2: Decision tree of depth 5 trained on simulated dummy variable data. Each internal node represents a decision rule based on one of three features: A, B, and C. The tree splits the dataset recursively based on these rules to separate "good" from "bad" outcomes. Each box contains the total samples contained at that node, the split of "good" vs "bad" as [Good, Bad], and the majority vote classification of each node. Gini coefficients are omitted for simpler visualization. Leaf nodes are the terminal points of the tree.

rules are encoded directly in the structure of the tree [23].

This structure is what makes decision trees so interpretable, as demonstrated in Figure 3.2. The decision path can be traced through the tree, showing exactly which decisions were made and in what order. If a prediction comes out as "bad," it is possible to trace back and conclude, for example, that the tickle delay was too short and the laser detuning was outside the optimal range. This level of interpretability is particularly valuable in experimental contexts, providing a concrete, rule-based explanation for why a configuration may be failing.

That said, decision trees also have drawbacks. One major issue is overfitting [24]. Since trees can keep splitting until every leaf is pure, they tend to memorize the training data —

capturing noise, outliers, or simulation artifacts that may not generalize [24]. A tree that grows too deep may detect spurious correlations that do not reflect true causal relationships, resulting in poor performance on unseen data [19].

Another challenge is instability [19]. Small changes in the training data — even the addition or removal of a single data point — can significantly alter the structure of the tree. This is due to the greedy, hierarchical nature of the splitting process [19, 20]. Consequently, decision trees can be sensitive and inconsistent, particularly when used in isolation. This instability is a key motivation for ensemble methods such as random forests, which average predictions across many trees to smooth out individual quirks [20].

There are also edge cases to handle — such as when a new data point falls in a region of the parameter space not previously encountered. Depending on the implementation, the tree may fall back to the nearest known leaf, interpolate among neighbors, or return a default prediction [17]. These edge-case behaviors become especially important in sparse or nonuniform datasets, as often encountered in simulation-driven work.

In practice, building a decision tree requires careful hyperparameter tuning — such as `max_depth`, `min_samples_split`, and `min_samples_leaf` — to balance model complexity and generalization [19, 22]. Even in its simplest form, however, a decision tree provides a remarkably transparent, rule-based model of the data [19]. It makes no assumptions about linearity, does not require feature scaling, and naturally accounts for parameter interactions — for example, if the effect of tickle frequency depends on trap voltage, the tree can capture that relationship by splitting accordingly.

3.1.1 Random Forests

While decision trees are intuitive and powerful, their very flexibility is also what makes them fragile. Even small changes to the training set can result in a radically different tree structure, due to the greedy, hierarchical nature of the splitting process. This instability and sensitivity pose challenges for generalization and model consistency. Random forests

are designed to address these issues directly [20, 22].

Rather than relying on a single decision tree, random forests build an ensemble — a collection of many decision trees, each trained slightly differently — and aggregate their predictions [22]. The general strategy is called bagging, short for bootstrap aggregation [22]. For each tree in the forest, a random sample of the training data is drawn with replacement (i.e., a bootstrap sample), a decision tree is trained on that sample, and the process is repeated multiple times [22]. Each tree thus encounters a slightly different version of the dataset — some points may appear multiple times, others not at all — which introduces beneficial diversity into the forest [22, 23].

This alone reduces variance [19].[5] Individual trees might make widely different predictions on a given input, but when their outputs are aggregated — typically by majority vote[6] in classification tasks — those outlier predictions are smoothed out [22]. Although any single tree may be inaccurate, the ensemble as a whole tends to be significantly more robust and accurate [22]. This effect is captured quantitatively by the variance of a random forest model:

$$\mathrm{Var}(\hat{f}_{\mathrm{RF}}) = \frac{\rho\sigma^2}{T} + (1-\rho)\sigma^2 \tag{3.3}$$

where σ^2 is the variance of an individual tree, ρ is the average correlation between trees, and T is the number of trees in the forest [22]. As T increases, the first term diminishes, reducing the overall variance.

Beyond this, random forests introduce randomness at the feature level [22, 23]. When each node considers a split, it does not examine all features in the dataset [23]. Instead, it selects a random subset — often $\sqrt{d}$, where d is the number of input variables — and identifies the best split among those [23].

This additional randomness prevents the trees from becoming too similar, particularly

[5] Bias refers to error from overly simplistic assumptions in the model (i.e., it underfits the data), while variance refers to error from sensitivity to small fluctuations in the training set (i.e., it overfits); the bias–variance tradeoff captures the balance between underfitting and overfitting that all models must strike [19].

[6] Majority vote means the most common class prediction among all trees is selected as the final output [19].

when a few dominant features might otherwise appear at the top of every tree. By encouraging the ensemble to explore less obvious partitions, the model becomes more effective at capturing nuanced patterns and generalizes better to unseen data [19, 23].

The end result is a model that is more stable, more accurate, and more resistant to overfitting than a single decision tree [20]. While some of the simplicity of a standalone tree is lost (it is no longer possible to walk through a single structure to extract a clear decision rule), random forests still provide substantial interpretability [20]. Feature importances — metrics that indicate how often and how effectively each parameter is used across all trees — are particularly informative. In the context of this thesis, such metrics offer insight into which simulation parameters most influence the classification task [19].

These rankings can be used not only for interpretation but also to guide future experiments. For example, if the forest consistently identifies cooling ramp time and tickle frequency as informative features, but finds beam alignment largely irrelevant, that provides a clear direction for optimization. Because the forest performs well even with relatively few training points — assuming the signal is sufficiently strong — it is particularly well-suited to scenarios where each simulation is costly and high information gain is essential.

Another valuable feature of random forests is out-of-bag (OOB) error estimation [19, 22]. Since each tree is trained on a different bootstrap sample, roughly one-third of the data is excluded from any individual tree's training. These unused data points serve as a built-in test set: for each one, predictions are averaged over only the trees that did not see it during training. This yields a reliable estimate of model accuracy without needing a separate validation set, which is especially convenient when simulation data is limited.

Some hyperparameters[7] still require tuning. The number of trees (`n_estimators`) affects both accuracy and computational cost — more trees generally improve performance but with diminishing returns. The parameters `max_depth`, `min_samples_split`, and `min_samples_leaf`

[7]Hyperparameters are user-specified settings that control the learning process, such as tree depth, number of estimators, or learning rate. Unlike model parameters, they are not learned from the data and must be set before training [19].

govern the complexity of individual trees [17]. Deeper trees can represent more complex interactions, but at increased risk of overfitting. Fortunately, random forests are more robust to such tuning compared to many other models, making them an appealing default in complex settings like this one [19, 20].

In summary, random forests provide a mechanism to retain the structural strengths of decision trees while overcoming their primary limitations — namely, high variance and sensitivity to noise. The result is a model that is robust, scalable, and informative [23]. In the context of high-dimensional, semi-noisy simulation output, random forests offer a compelling balance between predictive power and interpretability [23]. For this reason, they serve as a foundational component of the classification and analysis pipeline presented in this thesis.

3.1.2 Boosting

Boosting is a closely related technique, but with a key difference: rather than training trees independently, boosting trains them sequentially [24]. That is, instead of building a forest of trees in parallel and averaging their outputs as in bagging, boosting constructs a series of trees where each one is trained to correct the errors of the one before it [19, 24]. This makes boosting an inherently iterative method, where the model is refined step-by-step, homing in on the hardest-to-classify data points [24].

The process typically begins with a weak learner — usually a shallow decision tree — that performs only slightly better than random guessing [24, 25]. After training this first model, the algorithm identifies which data points were misclassified [24]. A second tree is then trained to correct those mistakes [24]. The subsequent tree focuses even more on the remaining misclassified points, and the sequence continues in this way. Over time, each successive tree contributes additional information about the more difficult edge cases [19].

The final prediction is a weighted combination of all trees in the sequence[8] — meaning that every tree contributes, but the more accurate ones have greater influence [20].

The most well-known variant of this method is Gradient Boosting, which formalizes the learning process through residuals [24, 26]. After each iteration, the model calculates the error — the residual between predicted and actual values — and fits the next tree to these residuals, effectively performing gradient descent in function space [23, 24]. This is the origin of the term "boosting": the performance of a weak model is improved through the directed, cumulative contribution of many such models [24]. Mathematically, the update step in gradient boosting can be written as:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{3.4}$$

as described in [24]. To illustrate this concretely: suppose the first tree classifies 70% of the simulations correctly, but struggles with a specific subset — such as configurations with low detuning and borderline cooling ramps. The second tree would be trained predominantly on those misclassified points, learning patterns that help distinguish them more accurately [23, 24]. The third tree would continue the refinement, improving the decision boundary where earlier models had difficulty. The result is a model that performs particularly well in capturing complex, nonlinear interactions, often achieving higher predictive accuracy than a random forest — especially when the signal is buried under noise or subtle in structure [24].

XGBoost, a widely used boosting algorithm, formalizes this optimization with the following objective function:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{3.5}$$

[8] In boosting, each tree's output is weighted based on its accuracy — better-performing trees contribute more to the final prediction, and the model aggregates these weighted outputs (e.g., by weighted majority vote or a weighted sum for regression) [19].

Here, ℓ is the loss function, f_t is the new tree added at iteration t, and $\Omega(f_t)$ is a regularization[9] term [19, 23, 24]. That regularization term is typically defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{3.6}$$

where T is the number of leaves, w_j is the score on each leaf, γ penalizes model complexity, and λ controls L2 regularization [19].[10]

This increased modeling power comes at a cost [22]. Boosting models — particularly when run for many iterations or with deep trees — are more prone to overfitting, especially when the dataset is small or noisy, as is often the case in simulation-driven contexts [23]. Because each tree is trained to fix the mistakes of its predecessor, there is a risk of fitting to noise unless the process is carefully regularized [19]. To mitigate this, boosting frameworks include parameters such as `learning_rate`, which governs how much each tree contributes to the final prediction, and `n_estimators`, which limits the total number of trees [24]. Smaller learning rates typically require more trees, but tend to yield better generalization by making updates more gradual [19, 24].

A related concept is shrinkage, which is simply another term for applying a reduced learning rate — intentionally damping the influence of each tree in the ensemble [19]. This slows training but often improves model performance. Additional techniques, such as subsampling (training each tree on a random subset of the data) and column sampling (selecting a random subset of features), introduce further randomness, which helps prevent overfitting and improves generalization — much like in random forests [19, 20].

From an implementation perspective, modern boosting libraries such as XGBoost, LightGBM, and CatBoost have significantly refined the core algorithm to make it faster, more memory-efficient, and more compatible with categorical or sparse data. These frameworks

[9] Regularization penalizes model complexity to prevent overfitting, often by discouraging overly large weights or deep trees [19].

[10] Also known as ridge regression, L2 regularization adds a penalty proportional to the square of model weights, encouraging smaller but nonzero values [19, 20].

incorporate strategies such as histogram-based binning and optimized split-finding to efficiently train models even on large datasets [22]. While this thesis focuses primarily on the scikit-learn implementation (`GradientBoostingClassifier`) for simplicity, these more advanced tools remain viable for future work aimed at further optimization.

For the purposes of this thesis, boosting offers a different perspective than bagging. While random forests are generally more stable and forgiving — well-suited for noisy or rugged parameter spaces — boosting can offer greater precision [19]. It is particularly adept at uncovering subtle structure and refining classification boundaries, though it requires more careful tuning and validation [19]. Both methods are therefore considered side-by-side in this work. Random forests provide reliable baselines and interpretable feature importance rankings, while boosting may yield sharper resolution in cases where random forests are less effective [19].

Ultimately, the choice between bagging and boosting is not a binary one. Each method addresses a different set of tradeoffs. Random forests are fast, interpretable, and highly usable out-of-the-box. Boosting offers a more refined approach that can deliver superior results when carefully tuned.[11] Given the structure of the problem at hand — high-dimensional, sometimes sparse, and often sensitive to small perturbations — incorporating both methods into the modeling pipeline is a natural and effective strategy.

3.2 Interpolation

Once a set of simulation results has been collected — parameter combinations and their corresponding cost function values — what remains is essentially a sparse, high-dimensional grid of sampled points. Each point represents a full simulation: a known configuration with a known outcome. However, there is no information about the vast majority of the parameter space that remains unsampled. To navigate it effectively, a method is required for estimating

[11] Boosting models are sensitive to hyperparameters like learning rate and tree depth; improper tuning can lead to overfitting or underperformance [19, 20].

outcomes at new, untested configurations. This is the role of interpolation.

At its core, interpolation refers to estimating values between known data points. For instance, if the outcomes of simulations are known at tickle frequencies of 420 and 430 kHz, it may be reasonable to infer something about the behavior at 425 kHz. In high-dimensional settings — especially when interactions between parameters are nonlinear or lack smoothness — this process becomes significantly more complex. In machine learning contexts, interpolation typically involves fitting a continuous model — such as a Gaussian process, a radial basis function (RBF) network, or a shallow neural network — that learns the shape of the cost surface and makes predictions not just at sampled points, but across the entire input domain [27, 28]. For example, an RBF model approximates the function as a weighted sum of localized basis functions:

$$\hat{f}_{\text{RBF}}(x) = \sum_{i=1}^{N} w_i \, \phi(\|x - x_i\|) \tag{3.7}$$

Meanwhile, Gaussian process regression defines a posterior mean function conditioned on prior observations [26, 27]:

$$\hat{f}_{\text{GP}}(x_*) = k(x_*, X) \, K^{-1} y \tag{3.8}$$

where $k(x_*, X)$ is the covariance vector between the new point x_* and training inputs X, K is the covariance matrix of X, and y is the vector of observed outputs [19, 23].

These models often include uncertainty estimates, which is particularly valuable in optimization: they indicate not only the predicted value of the cost function but also the confidence in that prediction.

It is important to distinguish interpolation from classification models such as random forests or boosted tree ensembles. While those models are designed to provide categorical decisions — for example, whether a configuration is "good" or "bad" — interpolation aims to reconstruct the continuous shape of the underlying function, often the cost surface, across

unsampled regions [27]. This makes interpolation particularly well-suited for gradient-free optimization, sensitivity analysis, or Monte Carlo sampling [19, 27]. Once a promising region is identified, interpolation enables a finer exploration of that space without requiring simulations at every possible point [19, 27].

In summary, classification models highlight which simulations warrant further attention, while interpolation models provide insight into the behavior between them. Both are integral to a complete optimization pipeline and, when used in combination, offer a far more comprehensive view of the parameter space than either method alone.

3.3 Monte Carlo Simulation

Monte Carlo simulation is another tool that fits naturally into this framework, though it operates somewhat differently from the other methods discussed thus far. It is not a learning algorithm — it does not build a predictive model or attempt to classify outcomes — but instead offers a probabilistic perspective for understanding uncertainty. In particular, it provides a principled way to explore how randomness in the system affects the outputs of interest, which is especially relevant in this case, where the simulation of scattering signals includes inherent variability [29].

The core idea of Monte Carlo methods is to approximate statistical properties of a system through repeated random sampling [19, 29]. In its most basic form, the Monte Carlo estimator of an expectation is written as:

$$\mathbb{E}[f(X)] \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{3.9}$$

where $x_i \sim p(x)$ are independent samples drawn from some distribution $p(x)$, and $f(x)$ is the function being averaged [29].

In the context of this thesis, Monte Carlo simulation is used to probe the stochastic variation in the photon scattering signal — which, even under fixed simulation parameters,

can fluctuate due to underlying probabilistic processes. Rather than assuming that each simulation produces a single, deterministic outcome, the output is treated as drawn from a distribution. By running multiple simulations with identical or slightly perturbed inputs and collecting the resulting scattering values, it is possible to empirically estimate the distribution of possible outcomes and, from that, derive error bars, confidence intervals, or variance estimates for each parameter configuration.

This becomes essential when evaluating whether a given configuration is consistently "good" — meaning it reliably yields a strong, slope-heavy signal — or if a favorable result occurred due to chance. Monte Carlo sampling enables a distinction between robust configurations and those that perform well only occasionally, which is critical for both classifier training and optimization strategies. It also provides a way to simulate the noise characteristics of a real experiment, where photon counts fluctuate between runs due to quantum shot noise, detection inefficiency, or thermal effects. Incorporating that variability into the model — rather than ignoring it — results in a learning process that is more robust and realistic.

Samples may be drawn uniformly across the parameter space, but are more commonly clustered around regions of interest — using biased sampling strategies to focus on neighborhoods near decision boundaries or where the cost function changes rapidly. This targeted approach allows for better resolution of subtle transitions and more accurate quantification of uncertainty where it matters most.

Thus, while Monte Carlo simulation does not perform classification or interpolation directly, it supports the two by enriching the dataset. It improves understanding of the stochastic structure of the simulation output, adds statistical weight to cost function evaluations, and ultimately facilitates more informed reasoning — not only about what the model predicts, but how [12] those predictions should be.

[12]Confidence here refers to the estimated reliability of a prediction, often quantified using statistical measures such as confidence intervals or empirical variability across repeated simulations [20].

3.4 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is the final algorithm covered in this thesis, and despite its simplicity, it is one of the most effective tools in the modeling pipeline. The idea behind KNN is conceptually straightforward: when classifying a new data point — for example, a new simulation whose outcome is not yet known — the algorithm examines the k closest examples from the existing dataset (where the outcomes are known) and assigns a label based on a majority vote [19]. If most of those nearby examples were "good" simulations, then the new one is likely to be good as well. If most were "bad," it is likely to be bad. No trees, no weights, no hidden layers — just distance and counting.

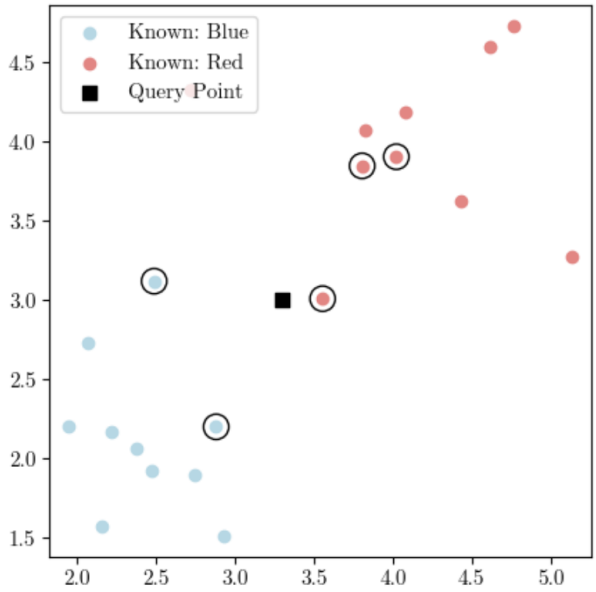


Figure 3.3: Simple KNN classification with $k = 5$. KNN classifies the black query point by examining its five nearest neighbors (circled). Since three are red and two are blue, the point is classified as red by majority vote.

What makes KNN powerful is that it operates entirely off the training data — there is no explicit "training" step. The algorithm does not build a model or fit parameters. Instead, it stores all the examples and performs computations only at prediction time. This makes KNN non-parametric and instance-based, meaning it adapts naturally to the local structure of the data, even in highly irregular or nonlinear regions [19, 20]. Distance is typically measured in

Euclidean terms (i.e., straight-line distance through parameter space), which is defined as:

$$d(x, x') = \sqrt{\sum_{i=1}^{n} (x_i - x'_i)^2} \tag{3.10}$$

but more tailored metrics, such as Manhattan distance,[13] can be used depending on the problem domain [19].

In cases where a weighted prediction is appropriate — for instance, when closer neighbors should influence the outcome more than distant ones — the following rule is used to compute a weighted average of neighbor outcomes:

$$\hat{y} = \left(\sum_{i=1}^{k} y_i / d(x, x_i)^2 \right) \Big/ \left(\sum_{i=1}^{k} 1 / d(x, x_i)^2 \right) \tag{3.11}$$

Here, y_i is the outcome of the i-th neighbor, $d(x, x_i)$ is the distance between the query point x and neighbor x_i, and closer neighbors contribute more heavily to the prediction [19].

In the context of this thesis, KNN is useful both as a lightweight classifier — to quickly assess whether a new parameter set resembles other successful configurations — and as a diagnostic tool. If KNN predictions begin to deviate significantly from those produced by more complex models, it may indicate that those models are overfitting, or that the local data structure exhibits higher complexity than anticipated [19]. In either case, KNN serves as a valuable sanity check — simple, direct, and effective when applied with care. Although KNN can be misleading if used improperly, it will be employed in ensemble with the other algorithms discussed to improve robustness and yield clean results [19, 23].

[13] Manhattan distance is the sum of the absolute differences between coordinates, like navigating a grid-like city.

3.5 Why not MLOOP?

When we first considered machine learning-based optimization for this system, a natural question arose: why not simply use MLOOP?[14] It's already integrated into QLICS, has a track record of success in other experimental platforms such as Bose–Einstein condensate production, and is explicitly designed to optimize physical systems using online Gaussian process regression [31]. In fact, our lab had already used MLOOP for basic parameter tuning tasks. So what changed?

The short answer is that while MLOOP is a powerful tool, it is not well suited to the specific goals of this thesis. MLOOP excels at locating a global minimum of a cost function using a relatively small number of evaluations [31]. It performs especially well when the cost surface is smooth, noise is moderate, and the parameter space is not excessively high-dimensional [30, 31]. Under these conditions, its hybrid approach — combining Gaussian processes with neural network-based sampling — allows it to converge quickly toward an optimal point [30]. This is ideal when the primary goal is to identify a single configuration that maximizes experimental performance [19].

However, that is not the goal here. This project is not about finding one good configuration — it's about understanding why certain configurations work at all. The focus is on building generalizable models that capture the structure of the parameter space and distinguish reliably successful regions from unstable or unreliable ones. In particular, this work aims to characterize the broader geometry of high-performing areas, not just their global extrema. After all, if we identify a single configuration with a very low cost but find that its surrounding neighborhood consists of poor outcomes, then any small drift in the hardware — a beam alignment shift, a voltage fluctuation — could push the system into failure. That isolated optimum might look promising on paper, but in practice, it's fragile.

This is the crux of the problem: optimization alone doesn't guarantee robustness. What

[14] MLOOP stands for Machine Learning Online Optimization Package — a toolkit for real-time experimental optimization using techniques like Gaussian process regression and evolutionary strategies [30, 31].

we really want are regions of parameter space that are both high-performing and stable — plateaus rather than sharp peaks [19, 30]. That requires more than just a minimum-finding algorithm. It requires a model that can learn the topology of the space, assess the density and quality of surrounding configurations, and generalize across nearby variations. MLOOP is not built for that kind of analysis.

Additionally, MLOOP offers limited interpretability. While it can propose new parameter combinations, it doesn't explain why those combinations are effective. It provides no insight into which parameters are most influential, how sensitive the outcome is to each one, or how the system responds to small perturbations. For a project focused not just on performance but also on physical understanding and downstream generalization, this is a fundamental limitation.

Thus, the framework developed in this thesis takes a different approach. Instead of treating the system as a black-box cost function to minimize, it frames the problem as a classification task: which configurations produce "good" outcomes, and why? It builds models that can interpolate across the parameter space, identify smooth high-performing regions, and expose which features matter most. The aim is not to replace MLOOP, but to answer a different question — one that depends as much on structure and stability as it does on optimization.

3.6 Summary

Each of the algorithms covered above — random forests, boosting, interpolation methods, Monte Carlo simulation, and KNN — offer different angles of attack on the same core problem: how can we use past simulation data to predict, classify, and ultimately optimize future experimental configurations? Each one comes with its own strengths and weaknesses, and their performance will depend on the structure and quality of the data we feed them. While this chapter has stayed grounded in theory, the following sections will tie these concepts di-

rectly to the trapped ion simulation framework — examining how each method contributes to improving number resolution, increasing dissociation accuracy, and reducing experimental trial-and-error in practice. That brings us to the next step: assembling these tools into a unified modeling framework. The following chapter outlines how each algorithm is implemented in practice, how they interact, and how their outputs guide the experimental optimization process from end to end.

Chapter 4

Cross-Algorithm Model

4.1 Framework Overview

In the previous chapter, we discussed the theoretical foundations behind various algorithmic tools used throughout this study. Having established that background, the goal of this chapter is to define the full architecture of our modeling framework — not only detailing the technical components of each algorithm, but also showing how they work together in practice. We provide a summary flowchart, a breakdown of model objectives, and representative outputs that illustrate how predictions and optimizations are made.

Before diving into implementation, it is essential to articulate the core objectives of the system:

1. To construct a robust predictive model capable of classifying any new parameter configuration as `good` or `bad`, while providing meaningful probability estimates and local stability metrics.

2. To guide the experimenter toward improved configurations using optimization paths

that are informed by both global learning and local geometric reasoning.[1]

These dual goals ensure that the system is useful not only for retrospective classification but also for prospective tuning and decision-making. The architecture is designed to answer two recurring experimental questions: *How good is this configuration?* and *What's the next best step I can take to improve it?*

To operationalize these goals, we first need to define how "goodness" is quantified. This is accomplished via a scalar *cost function*, which evaluates the quality of each configuration based on its dissociation behavior — particularly its ability to support clear number resolution across successive scattering transitions. As previously discussed, this resolution is critical for tracking dissociation progression and measuring μ over time. Therefore, the initial cost function should focus on numerical signal quality — for instance, penalizing shallow slopes or non-monotonic scattering behavior.

However, the key insight is that there is no universal cost function suitable for all experimental goals. Depending on the context, one might prioritize sharp transitions, robustness to noise, or consistent slope across all steps. For this reason, the cost function must remain flexible and user-defined. Later results will explore several candidate cost functions that reflect different priorities: some are strictly derived from features of the scattering trace (e.g., slope and R^2), while others include probabilistic and stability-aware modifications. For now, we recommend the following three-step strategy for practical implementation:

1. **Define a base cost function.** This should quantify the sharpness and consistency of the scattering trace, and will typically combine normalized slope and R^2 values. As we will show in future sections, these features can be normalized[2] to lie within a fixed range (e.g., [0, 100]) to support consistent comparisons across the dataset.

[1] Combining global learning with local geometric reasoning enhances optimization efficiency by balancing broad exploration with precise exploitation. This synergy has proven effective in complex optimization tasks, as demonstrated by Zhou et al., who integrated global and local surrogate models to accelerate evolutionary optimization [32].

[2] Normalization refers to the process of rescaling numerical values to a common range, often [0, 1] or [0, 100], to allow consistent comparison across different features or units.

2. **Apply percentile-based classification.** To assign binary labels, we recommend choosing a percentile cutoff based on the cost distribution — for example, classifying the top 50% (lowest cost) as `good` and the rest as `bad`. While the exact split can be tuned later, using a percentile-based method ensures that initial labels reflect the relative quality of configurations, rather than relying on arbitrary thresholds.

3. **Define an extended cost wrapper**[3]. This secondary cost function can incorporate additional terms, such as Monte Carlo-based uncertainty or local KNN stability. These modifiers weight the base cost function based on how robust or trustworthy a configuration is under repeated trials or in its surrounding neighborhood. This layered approach reflects the reality that experimental priorities shift over time — and that not all "good" configurations are equally reliable.

This structured but adaptable framework ensures that classification decisions are grounded in physical observables, while still allowing for flexibility and refinement as experimental goals evolve. In the following sections, we outline the model layers that make this functionality possible, including supervised learning, Monte Carlo variance modeling, and local interpolation-based robustness scoring.

Now that we have covered the theoretical background of the algorithms we incorporate in Chapter 3 and laid out the basic framework of our model, we can dive into the technical details:

4.2 Model Architecture and Functionality

First, we must explicitly define what we mean by "build a model." In prior sections, we have described a machine learning algorithm as a dynamic learner — conceptually, a new lab partner being onboarded to understand and navigate the experiment alongside us. Building

[3]A wrapper here refers to a higher-level function that takes the base cost and modifies it with additional terms or conditions, effectively "wrapping" extra logic around the original calculation.

on this analogy, our high-level objective is to develop a model that captures the full dynamics of the system and enables targeted exploration of the experimental feature space. To achieve this, we require not only a machine learning model capable of accurate classification, but also the incorporation of statistical methodologies that account for uncertainty and local stability within different regions of the parameter space. The complete system is composed of three tightly integrated components: (1) a trained classification model (Random Forest or XGBoost) that learns global structure from simulation data, (2) a Monte Carlo extension that captures stochastic variability across repeated trials, and (3) a local stability framework based on K-Nearest Neighbors (KNN) and Radial Basis Function (RBF) interpolation. Together, these layers form a probabilistic, interpretable, and highly adaptable decision engine — one that supports both evaluation and optimization of experimental configurations in real time.

4.2.1 Classification Model

We use both Random Forest and XGBoost classifiers in tandem, as each brings distinct advantages: Random Forests reduce variance through bagging, while XGBoost minimizes bias via sequential boosting. The dataset is split into 80% training and 20% testing sets,[4] and both models are trained on identical features — RF drive voltage, endcap voltage, and modulation amplitude — with binary labels derived from cost-function-based thresholds. Hyperparameters (`max_depth=6`, `n_estimators=300`, `min_samples_leaf=4` for Random Forest; `learning_rate=0.1`, `max_depth=5`, `n_estimators=250` for XGBoost) were selected via a combination of grid search and randomized search to maximize F1-score on a subset of validation data. Either model may be deployed depending on the user's priorities, and both are seamlessly integrated into the broader system pipeline.

[4]This is a standard practice for most models to ensure sufficient data for training while preserving a representative holdout set for evaluation.

4.2.2 Monte Carlo Extension

To account for stochastic variability in the simulation environment, we run each parameter configuration multiple times (with randomized initial conditions inherently incorporated in QLICS and LAMMPS). This yields an empirical distribution of outcomes for each setup, from which we compute mean values, standard deviations, and class probabilities. These repeated trials allow the model to assign confidence estimates to predictions and identify configurations with unstable or unreliable behavior. All Monte Carlo outputs are propagated through the classification and cost function pipelines, enabling uncertainty-aware optimization and filtering. This approach avoids any assumptions about the shape of the underlying distribution and instead relies purely on empirical variance.

4.2.3 KNN-Based Local Interpolation

We implement a local manifold-based interpolation framework that intelligently combines multiple components to evaluate the robustness of a given configuration. This method grounds itself in real, observed data using K-Nearest Neighbors (KNN), constructs geometric shells (spheres) around a query point to enable structured, distance-based spatial sampling, and applies Radial Basis Function (RBF) interpolation to estimate both class labels and cost values at interpolated points on those shells. By integrating these elements, the method defines a neighborhood-informed probability score that reflects how robust or stable a given configuration is likely to be in its local region of the parameter space — a principled, geometric, and fully implementable approach for quantifying class robustness under uncertainty.

The implementation proceeds as follows. First, all parameter data is standardized,[5] and a `NearestNeighbors` model is fit to the scaled dataset. Two `RBFInterpolator` models are then trained — one on binary class labels (mapped to 0 and 1) and another on scalar cost values

[5]That is, each feature is rescaled to have zero mean and unit variance, so all input dimensions are on the same scale.

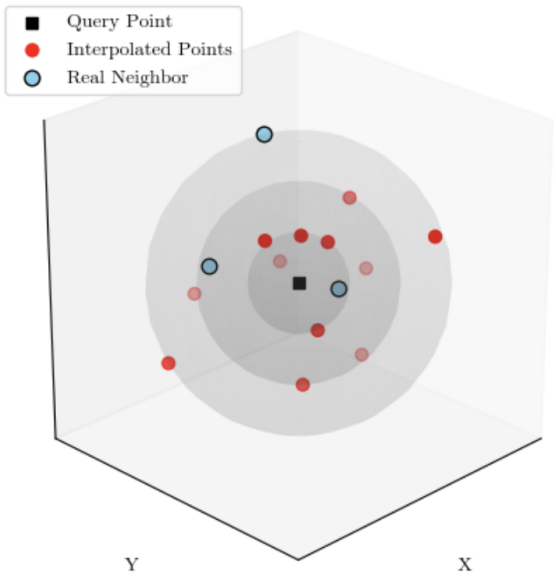


Figure 4.1: A visual depiction of the local manifold-based interpolation framework. A query point (black square) is surrounded by three concentric spheres, each centered at increasing radii based on real neighbors (blue dots). For each radius, four additional points (red dots) are uniformly sampled on the sphere surface and classified using RBF interpolation. This results in five total points per shell — one real, four synthetic — and fifteen points in total. These local samples are used to compute a good-to-bad ratio, quantifying the query point's stability within its neighborhood.

— using thin-plate spline kernels.[6] For a given query point, we identify its three nearest neighbors with unique distances, construct hyperspheres of equal radii around the query, and randomly sample synthetic points on each shell, as shown in Figure 4.1. The class labels of these synthetic points are then estimated using the RBF interpolation model and combined with the real neighbors to compute a good-to-bad ratio, defined as the number of "good" classifications divided by the number of "bad" classifications among the local neighbors — that is, Good-to-Bad Ratio = #Good/#Bad. This score serves as a quantitative measure of local stability: points with many "good" interpolated neighbors are considered stable, while those near boundaries yield lower scores. The result is an interpretable, data-driven metric that complements global model predictions and Monte Carlo uncertainty estimates,

[6]The thin-plate spline kernel is a smooth, radially symmetric function that minimizes bending energy, making it well-suited for interpolation tasks requiring smooth surfaces across multidimensional space [33].

enhancing our ability to assess configuration quality in both absolute and relative terms.

4.3 Unified Output and Example

When these three systems are combined, the result is a robust and highly informative modeling framework.[7] For any given parameter configuration, the system outputs:

→ A binary classification (`good` or `bad`)

→ A classification probability based on Monte Carlo variance estimates of key metrics

→ A KNN-based stability score using interpolated neighbor sampling

Beyond static evaluation, the model also includes a `find_shortest_path_to_good` routine. This identifies the closest `good` configuration in the dataset (using a KD-tree),[8] ranks the parameter changes needed to reach it based on feature importance, and recommends a minimal adjustment sequence. Continuous interpolation along this path can be used to estimate success probabilities along intermediate points, supporting gradient-aware exploration.

Together, these tools provide an end-to-end workflow for simulation-informed experimental control. The model classifies new configurations, evaluates how confidently they succeed, estimates their local robustness, and proposes optimal adjustments to move toward better performance — all while accounting for uncertainty and system stochasticity. This multi-tiered structure enables actionable insights, even in the presence of noise and nonlinear parameter interactions.

To evaluate the quality of each simulation trace, we define an initial composite cost function based on two features: the slope of the dissociation trace and the R^2 value of a linear fit to that trace.[9] A steep negative slope is desirable, as it indicates a sharp drop in signal — a

[7] Corresponding Python code for these models will soon be posted on my GitHub page.

[8] A KD-tree (k-dimensional tree) is a spatial data structure used to efficiently find nearest neighbors in multi-dimensional space [19, 34].

[9] Each trace is composed of the sequential scattering values (photon counts), thus the line of best fit is fit through these points using simple OLS regression [19].

hallmark of strong motional coupling and thus a clearer dissociation signature. Meanwhile, a high R^2 value reflects a consistent, monotonic trend rather than random fluctuations, suggesting that the system behavior is coherent rather than noise-dominated.

To combine these two features into a unified score, both must be normalized. The slope is normalized such that the most negative slope — representing the strongest signal — is assigned a value of 0, and the most positive slope (i.e., the weakest or inverted signal) is assigned a value of 100. This ensures that lower values correspond to better experimental outcomes. The normalization is defined as:

$$\texttt{Slope\_norm} = \left(\frac{\texttt{Slope} - \texttt{min\_slope}}{\texttt{max\_slope} - \texttt{min\_slope}} \right) \cdot 100, \tag{4.1}$$

where `min_slope` and `max_slope` denote the most negative and most positive slope values in the dataset, respectively.

The R^2 value is treated inversely: since higher values are better, we subtract each from 1 and scale accordingly:

$$\texttt{R-sq\_norm} = (1 - \texttt{R-sq}) \cdot 100. \tag{4.2}$$

This transformation ensures that, like the slope, lower normalized R^2 values correspond to better fits, aligning the two features under a common optimization direction.

Finally, the normalized features are combined into a single cost function via a weighted sum, emphasizing the slope more heavily (80%) while still incorporating trace consistency (20%). The full cost function is:

$$\text{Cost} = 0.8 \cdot \texttt{Slope\_norm} + 0.2 \cdot \texttt{R-sq\_norm}. \tag{4.3}$$

This formulation balances the sharpness and reliability of the dissociation signal, providing a meaningful scalar target for classification and optimization tasks throughout the modeling pipeline. The decision to weight the slope more heavily stems from its greater direct relevance

to dissociation detection: a steep slope reflects the physical transition we aim to resolve, whereas a high R^2 simply ensures that the trend is not noisy or erratic.

It is important to emphasize that this is a preliminary cost function — simple and interpretable, but not exhaustive. Future sections will introduce more nuanced formulations that incorporate uncertainty quantification, multi-stage behavior, and KNN-based stability scores. These richer cost functions aim to better capture the complexity of dissociation behavior in the presence of noise and variability. For the current modeling stage, however, this 80/20 formulation offers a transparent, intuitive starting point that aligns well with our physical expectations.

Using the cost function, we now define a binary classification task to train our supervised machine learning model. Specifically, we set a threshold at the 20^{th} percentile of all cost values — a conservative cutoff that isolates the top-performing 20% of configurations as `good`. All remaining simulations are labeled `bad`.[10]

Suppose now that we test our two-part model on the parameter set `(Voltage = 58.9, Endcap Voltage = 2.0, Modulation Amplitude = 0.8)`. The classification model predicts this configuration to be classified as `bad`, with a high estimated probability of 0.972 that it remains `bad` under stochastic perturbations derived from empirical noise. The interpolated neighbor good-to-bad score is also low at 0.07, further indicating that this configuration is locally unstable with respect to improvement.[11]

To improve the system, the model recommends transitioning to the nearest known `good` configuration, which is `(Voltage = 58.7, Endcap Voltage = 2.0, Modulation Amplitude = 0.8)`, as shown in the flowchart below. The only suggested adjustment is a minor decrease in `Voltage` from 58.9 to 58.7, while keeping the other two parameters fixed. This adjustment is derived based on the relative feature importances detailed in Section 3.1.1, prioritizing changes to the most influential parameter first.

[10] 20th Percentile Cutoff for Cost Function in this case: 32.3638.

[11] In this context, "improvement" refers to the likelihood that small, local parameter changes might yield a better (i.e., "good") outcome.

Input Configuration

Voltage = 58.9, Endcap Voltage = 2.0, Modulation Amplitude = 0.8

Predicted classification: `bad`

Estimated probability it stays classified as bad: 0.972

Interpolated neighbor good-to-bad score: 0.07

⇓

Recommended nearest `good` configuration: Voltage = 58.7, Endcap Voltage = 2.0, Modulation Amplitude = 0.8

Change `Voltage` from 58.9 → 58.7

⇓

New Configuration

Voltage = 58.7, Endcap Voltage = 2.0, Modulation Amplitude = 0.8

Predicted classification: `good`

Estimated probability it stays classified as good: 0.884

Interpolated neighbor good-to-bad score: 0.15

Evaluating this new configuration, the model predicts it as `good`, with an estimated probability of 0.884 that it remains classified as `good` under empirical noise perturbations. This is a strong result — a high probability that the system will maintain desirable behavior despite stochastic fluctuations. However, the interpolated neighbor good-to-bad score remains low at 0.15, suggesting that the configuration lies in a fragile region of parameter space: while it performs well in isolation, its immediate neighbors are mostly `bad`. This contrast highlights the dual role of our two-part model — assessing both robustness to noise and structural

resilience.

This raises a natural question: what physical properties of the system make it so sensitive that even a slight voltage change can produce such a dramatic shift in classification? This kind of instability often arises from the sharpness of motional resonances in the system. Near a secular frequency, even a small change in trap voltage can shift the resonance condition dramatically, either aligning modulation with the motional mode — enhancing dissociation visibility — or missing it entirely, resulting in a flat or noisy trace. These transitions are not gradual: due to the steep frequency response of the crystal to modulation when coupling is strong, the cost landscape can exhibit rapid drops or jumps in quality over narrow voltage intervals.[12] In this case, the steep transition likely corresponds to a narrow resonance window, where constructive motional excitation temporarily improves signal clarity. Slightly detuning from this window — by adjusting the RF amplitude or endcap voltage by even a fraction of a volt — can suppress this excitation and degrade number resolution, leading to sharp classification boundaries.

To interpret this situation constructively, we consider how different model outputs can be brought together to guide user decisions. Essentially, what the above result tells us is that even though this move places us in a `good` configuration, it may still lie near a decision boundary — a region where small, unmodeled perturbations could lead to reclassification or degraded performance. In such cases, and as hinted in earlier discussions of cost design, it may be advisable to re-evaluate the initial cost function or introduce a higher-level cost wrapper that integrates additional model outputs. Recall that the user retains full control over what qualifies as "good," and can manually adjust thresholds or explore more conservative regions of parameter space. One way to implement this would be to modify the cost function to reward not only strong slope and R^2 values, but also high neighbor stability scores — thereby prioritizing configurations that perform well *and* lie within robust neighborhoods.

To incorporate neighbor stability into the cost in a nonlinear way, we introduce a multi-

[12] We demonstrate this via scattering heatmap in Section 5.2.1.

plicative weighting scheme that penalizes instability rather than simply summing costs. The original cost function remains:

$$\texttt{Slope\_norm} = \left(\frac{\texttt{Slope} - \texttt{min\_slope}}{\texttt{max\_slope} - \texttt{min\_slope}} \right) \cdot 100, \tag{4.4}$$

$$\texttt{R-sq\_norm} = (1 - \texttt{R-sq}) \cdot 100, \tag{4.5}$$

$$\text{Base\_Cost} = 0.8 \cdot \texttt{Slope\_norm} + 0.2 \cdot \texttt{R-sq\_norm}. \tag{4.6}$$

We then extend this formulation to incorporate a neighborhood sensitivity term. To account for neighborhood robustness, we introduce a multiplicative penalty term based on the interpolated good-to-bad score. Because higher scores represent stronger local support, we invert the stability score to penalize fragile configurations. The final wrapped cost becomes:

$$\text{Wrapped\_Cost} = \frac{\text{Base\_Cost}}{1 + \alpha \cdot \texttt{Stability\_score}}, \tag{4.7}$$

where α is a tunable weighting parameter that controls how strongly the neighborhood score impacts the final cost (e.g., $\alpha = 1$). This formulation ensures that as the stability score increases, the total cost decreases nonlinearly — amplifying the value of configurations that are both performant and robust to local perturbations. For example, if the base cost is 70 and the neighbor score is 2.0, the final cost would be:

$$\text{Wrapped\_Cost} = \frac{70}{1 + 2.0} = 23.33,$$

reflecting a highly favorable configuration. Conversely, unstable points (with neighbor scores near 0) retain their full base cost or worse, discouraging the model from selecting fragile configurations.

This cost wrapper, while simple, allows the experimenter to balance fit quality with

stability — a key consideration when transitioning from exploratory modeling to physical implementation. By surfacing both classification and neighborhood risk, the model offers a level of interpretability and flexibility that would otherwise be difficult to achieve. This process makes the search for optimal configurations more transparent and principled, allowing for informed refinement toward not only high-performing, but also stable and reliable outcomes.

Taken together, these tools form a flexible, modular architecture for navigating the feature space of trapped ion experiments. By combining classification, probabilistic scoring, and local interpolation, the model not only identifies promising configurations, but also quantifies their surrounding risk — enabling smarter, more informed optimization than blind trial-and-error.

With the modeling infrastructure in place, we now shift our focus to the empirical results. The next chapter begins by analyzing key physical patterns that emerge from the simulation data itself, independent of any machine learning, before moving on to a full performance evaluation of the modeling framework introduced here.

Chapter 5

Results

This chapter presents the major findings of this study, structured into two complementary parts. We begin by analyzing the raw simulation data from a deterministic perspective, identifying core physical trends in dissociation behavior, number resolution, and parameter dependencies. We then extend this analysis by incorporating stochasticity through Monte Carlo methods — repeatedly simulating the same configuration to model variability and establish a distribution over possible outcomes. This allows us to identify not only average performance, but also the robustness and reliability of each configuration under noise. Based on this framework, we suggest candidate solutions that balance performance with stability. In the second part of the chapter, we turn to the machine learning models themselves — evaluating their predictive accuracy, classification fidelity, and practical utility in proposing new configurations. Although distinct in focus, these two parts form an integrated loop: the physics insights inform model design, and the model results, in turn, validate and refine our physical understanding.

5.1 Logistics

It is helpful to conceptually separate the problem of detecting the first dissociation event from the broader challenge of achieving full number resolution. The first step — going from

zero to one O^+ ion in the trap — is not necessarily easier in a signal-to-noise sense, but it is the most logically constrained and experimentally accessible starting point. When no O^+ ions are present, the background is as clean as it will ever be: any observed modulation-induced change in Be^+ fluorescence must be attributable to the presence of a newly formed O^+ ion. This makes the first step a useful benchmark. It allows us to validate our dissociation protocol, test whether modulation at the expected secular frequency causes a measurable change in the signal, and tune core parameters like modulation amplitude and trap voltages in a relatively low-complexity regime. If we can't reliably detect the formation of the first O^+ ion, then attempting to distinguish two from three — or six from seven — is premature. So while the first step isn't inherently more visible, it serves as the foundation for everything that follows.

Once we've established that the system can detect the onset of dissociation, the challenge becomes finer-grained: can we resolve how many O^+ ions are present? This is what we mean by *number resolution* — the ability to distinguish between different O^+ counts based on the structure of the Be^+ fluorescence trace. Unlike the binary jump from zero to one, these higher-number transitions are subtler. Each additional O^+ ion produces a smaller marginal effect on the motional coupling and, in turn, the Be^+ fluorescence signal. The slope of the dissociation trace may still change, but more gradually, and with greater sensitivity to thermal noise, trap alignment, and subtle parameter interactions. In this regime, simple heuristics break down, and our models must learn to identify not just the presence of a signal, but structured variation across a noisy, high-dimensional space. Achieving reliable number resolution is essential for extracting quantitative information about dissociation dynamics — and by extension, for improving the precision of molecular transition measurements. It's the difference between saying "something happened" and saying "exactly this many dissociation events occurred," which is critical for the spectroscopy goals that motivate the experiment.

With this motivation established, we now turn to the raw simulation outputs to identify how dissociation behavior emerges, how it is shaped by control parameters, and what this

reveals about the physical structure of the system. To build a comprehensive understanding, we begin by examining the physical structure of the data itself — before any learning occurs.

5.2 Physics Results

Throughout the following sections, we present results in a logical and chronological manner. Therefore, it is important to begin with our starting point in Figure 5.1.

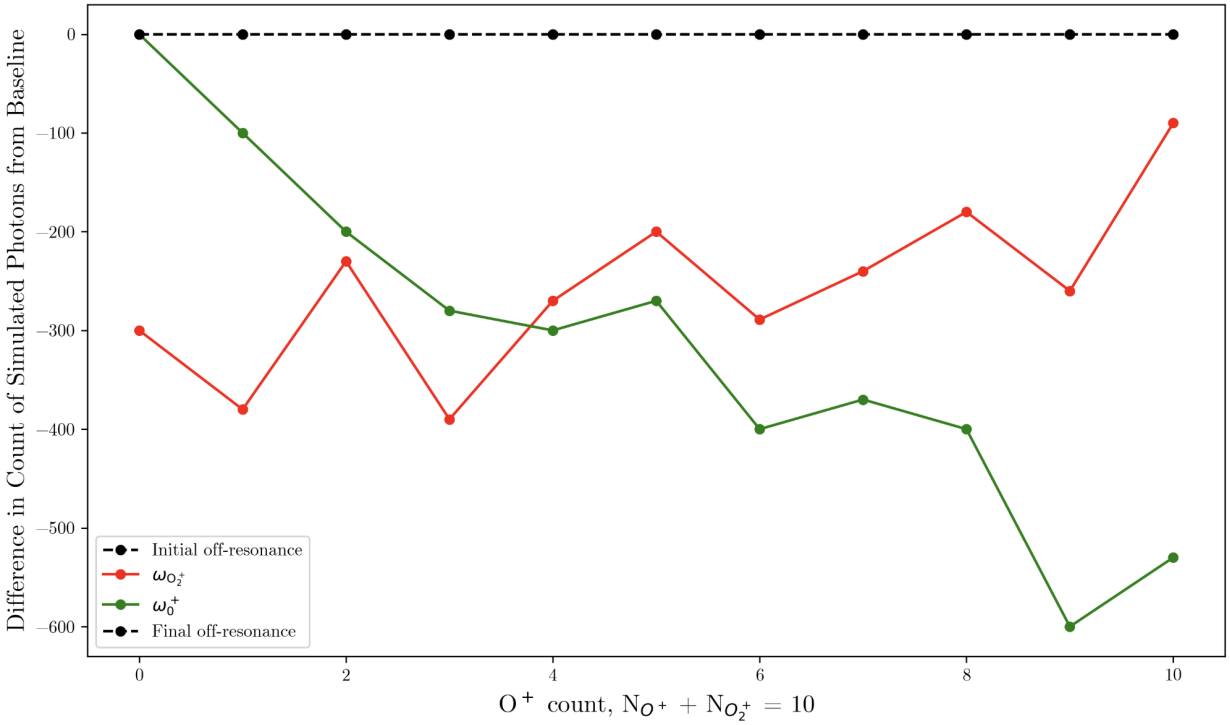


Figure 5.1: Number resolution for a total of ten ions in the trap, where green represents the count of O^+ and red represents the count of molecular oxygen as compared to an off-resonance baseline in black. Note also how O^+ offers much better number resolution than O_2^+, due to its mass being more similar to Be^+. Due to this property, most analysis will be performed on the atomic oxygen signal rather than the molecular oxygen signal [3].

To better understand the contribution of each species to the overall resolution pattern, we isolate the atomic oxygen signal and examine it independently. When we isolate the O^+ signal from Figure 5.1, we are left with the plot in Figure 5.2.

With a general sense of the dissociation signal in hand, we now zoom into the most

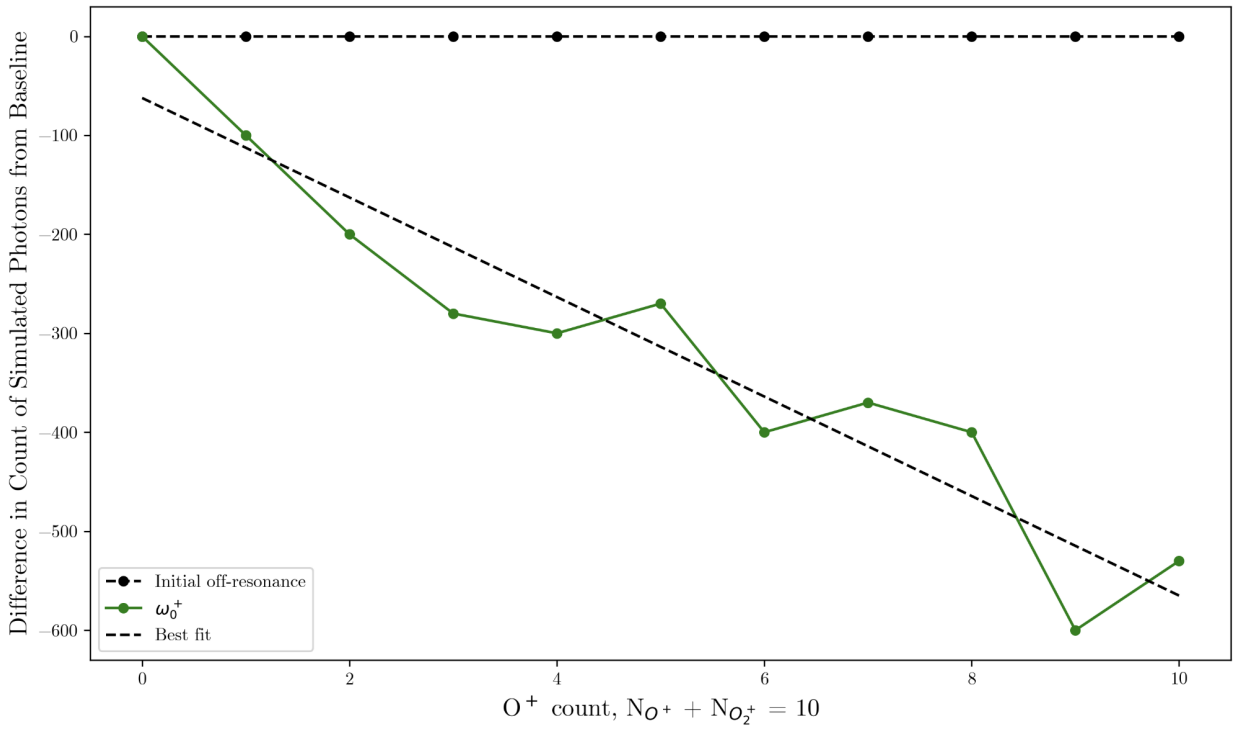


Figure 5.2: Isolated atomic oxygen signal from Figure 5.1 with a fitted line of best fit. Although the line of fit itself has a significant negative slope and the points are clustered relatively closely to the line, note that the actual number resolution is poor. Try to differentiate the signal between 3 O^+ and 5 O^+, for instance, or between 6 and 8 O^+ in the trap. Though overall this example appears solid, these crucial flaws represent the resolution we wish to gain through our analysis.

fundamental transition: the detection of a single dissociation event.

5.2.1 Single Dissociation Step

As we turn to the single dissociation step, in which we aim to detect the first dissociation event from no O^+ to one O^+ in the trap, we must explicitly define what a `good` outcome is. Since we simply wish the minimize the scattering signal as much as possible to resolve it from the baseline, we base `good` on an arbitrary cutoff scattering value. Essentially, classification is based upon a simple threshold function, where all values above cutoff c are classified as

"bad" and all values below c are labeled as "good" outcomes.[1] Note that the thing being classified, as mentioned throughout Chapter 4, is the set of parameters corresponding to the outcome, not the outcome itself.

As discussed in Section 3.1.1, a crucial advantage of Random Forest models is their ability to generate a feature importance scale which lists explicit percentage values of "how important" each feature in the model was for classification (more information in Section 3.1.1). Instead of blindly sampling the entire parameter space of potential configurations, we conducted initial Random Forest classification to determine which features to sample most closely. The resulting model indicated that voltage, endcap voltage, and modulation amplitude were the primary drivers of scattering signal decrease among all possible features.[2]

This data-driven insight aligns well with theoretical expectations from ion trap physics. Recall that the RF and endcap voltages together define the radial and axial confinement of the trap, and thus determine the overall shape and spatial extent of the ion crystal. A higher endcap voltage, for instance, compresses the crystal along the trap axis, while increased RF amplitude strengthens radial confinement [3]. These voltage settings directly influence the equilibrium positions of ions, particularly the spatial separation between species with different charge-to-mass ratios. Because dissociation is detected indirectly — via motional coupling between O^+ ions and the laser-cooled Be^+ — the geometry of the crystal becomes critically important. When O^+ ions are tightly confined and positioned closer to the Be^+ core, their motion perturbs the Be^+ ions more strongly, producing a clearer fluorescence signal. Conversely, if the crystal is too elongated or the species too far apart, this coupling weakens, and dissociation events become harder to detect. Optimizing the trap voltages is therefore a key lever for maximizing the signal contrast associated with single or multiple dissociation events. In addition to the trap voltages, the modulation amplitude is another

[1] This differs from the percentile-based cutoff used in Chapter 4. Here, since each simulation yields only a single scattering value — which directly represents the cost — applying a fixed numerical threshold is equivalent in practice to using a percentile, and simplifies evaluation.

[2] Here, "all possible features" refers to the quantities listed in the annotated configuration file used for simulation input.

crucial parameter for dissociation detection. A sufficiently large modulation amplitude is required to drive detectable motion in the O^+ ions, which in turn perturbs the Be^+ ions through Coulomb coupling. However, if the amplitude is too high, it can destabilize the crystal or blur the dissociation signal, so it must be carefully tuned to maximize sensitivity without degrading signal fidelity.

To probe this connection more directly, we fix the majority of parameters and conduct a controlled sweep over the trap voltages. We generate configuration files over a range of plausible RF voltages and endcap voltages[3] , while holding the modulation frequency and amplitude fixed. As discussed in Section 1.3.1, the modulation frequency is a function of both the RF and endcap voltages, as it depends on the resulting secular frequencies of the trapped ions. Because resonant excitation occurs only when the modulation frequency matches the corrected radial secular frequency,

$$\omega_{x,y} = \sqrt{\omega_r^2 - \frac{\omega_z^2}{2}}, \tag{5.1}$$

only configurations that satisfy this resonance condition will result in strong motional coupling and, consequently, minimal Be^+ fluorescence due to Doppler broadening [1, 5]. This means the modulation frequency must be precisely matched — or "dead on" — to the secular frequency of the target species (O^+) for scattering to reach a minimum. We fix the modulation frequency at 387000 kHz rather than calculating it for each individual parameter combination. Given that the lowest scattering is achieved on-resonance, we expect to see a narrow ridge of low-scattering points across the voltage–endcap grid. Even slight deviations in voltage or endcap voltage shift the secular frequency enough to significantly diminish the resonance effect, causing scattering to increase. This sharp dependence makes an observed ridge both a diagnostic tool and a tuning aid: it enables fine-grained adjustment of the trap configuration to align the modulation frequency with the system's natural motional

[3]These ranges are determined based on the experimentally validated stability curve shown in Wolfgang Paul's paper: [35].

frequencies.

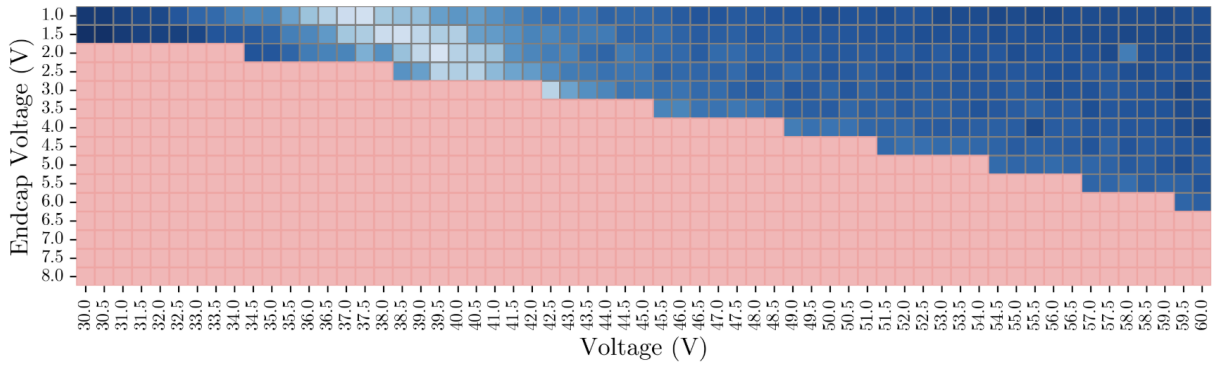


Figure 5.3: Heatmap visualization of scattering signal for various combinations of voltage and endcap voltage. In this heatmap, darker shades of blue represent scattering values increasingly close to the baseline, whereas values that are very light (or even white) correspond to the largest decreases in photon count from the baseline. Note that pink values correspond to simulations that threw errors, i.e. trap parameters violated stability conditions [35].

Figure 5.3 encapsulates the correct relationship: indeed, only a specific "streak" of voltage-endcap voltage pairs results in the lowest scattering signal. However, notice that this white streak corresponds to configurations in the range of a voltage of 39.0 V and endcap voltage of 2.0 V.

Suppose we calculate the resonance frequency corresponding to a voltage of 39.5 V and endcap voltage of 2.0 V, values which correspond to one of the lowest (most white) scattering signals in Figure 5.3. Note that the RF Voltage (shown along the x-axis) is the peak-to-peak value, and thus we must halve it for input into the expression for q, so we set $V_0 = \frac{39.5}{2} = 19.75\,\text{V}$.

First, from equations drawn from the introductory chapter, we compute the q-parameter:

$$q = \frac{2e(2V_0)}{mr_0^2\omega^2} = \frac{2 \cdot (1.602 \times 10^{-19}) \cdot 2 \cdot 19.75}{(2.66 \times 10^{-26}) \cdot (0.00125)^2 \cdot (2\pi \cdot 11.04 \times 10^6)^2} \Rightarrow q \approx 0.0633$$

Next, the radial secular frequency ω_r is:

$$\omega_r = \frac{q\omega}{2\sqrt{2}} = \frac{0.1407 \cdot 2\pi \cdot 11.04 \times 10^6}{2\sqrt{2}} \Rightarrow \omega_r \approx 1.552 \times 10^6 \,\text{rad/s}$$

The axial frequency ω_z is calculated using:

$$\omega_z = \sqrt{\frac{2\kappa eU}{mz_0^2}} = \sqrt{\frac{2 \cdot 0.17 \cdot 1.602 \times 10^{-19} \cdot 2.0}{2.66 \times 10^{-26} \cdot (0.0015)^2}} \Rightarrow \omega_z \approx 1.349 \times 10^6 \,\text{rad/s}$$

The corrected transverse secular frequency ω_{xy} becomes:

$$\omega_{xy} = \sqrt{\omega_r^2 - \frac{1}{2}\omega_z^2} = \sqrt{(1.552 \times 10^6)^2 - \frac{1}{2}(1.349 \times 10^6)^2} \Rightarrow \omega_{xy} \approx 1.224 \times 10^6 \,\text{rad/s}$$

Finally, converting to kHz:

$$f_{\text{final, corrected}} = \frac{f_{xy}}{2\pi} = \frac{1.224 \times 10^6}{2\pi} \Rightarrow f_{\text{final, corrected}} \approx 194.8 \,\text{kHz}$$

Interestingly, the calculated frequency deviates significantly from the modulation frequency used in the simulations (roughly 387 kHz). In fact, the theoretical value at which we would expect the greatest dip in scattering signal is nearly precisely half of the utilized value. Perhaps we are missing a larger trend: looking more broadly at the scattering signal while sweeping over a larger range of frequencies, we get the result in Figure 5.4.

Although there is indeed a dip in the scattering signal at the theoretical resonant frequency, we in fact see a much larger dip at *twice* this frequency. This suggests that while the system exhibits behavior indicating that it iss being modulated near the expected secular frequency of motion ($\sim 195\,\text{kHz}$), the most prominent effect on Be^+ fluorescence — and thus the most visible "detection" of motion — is occurring at a higher harmonic, specifically at $\sim 390\,\text{kHz}$, nearly twice the calculated $\omega_{xy}/2\pi$.

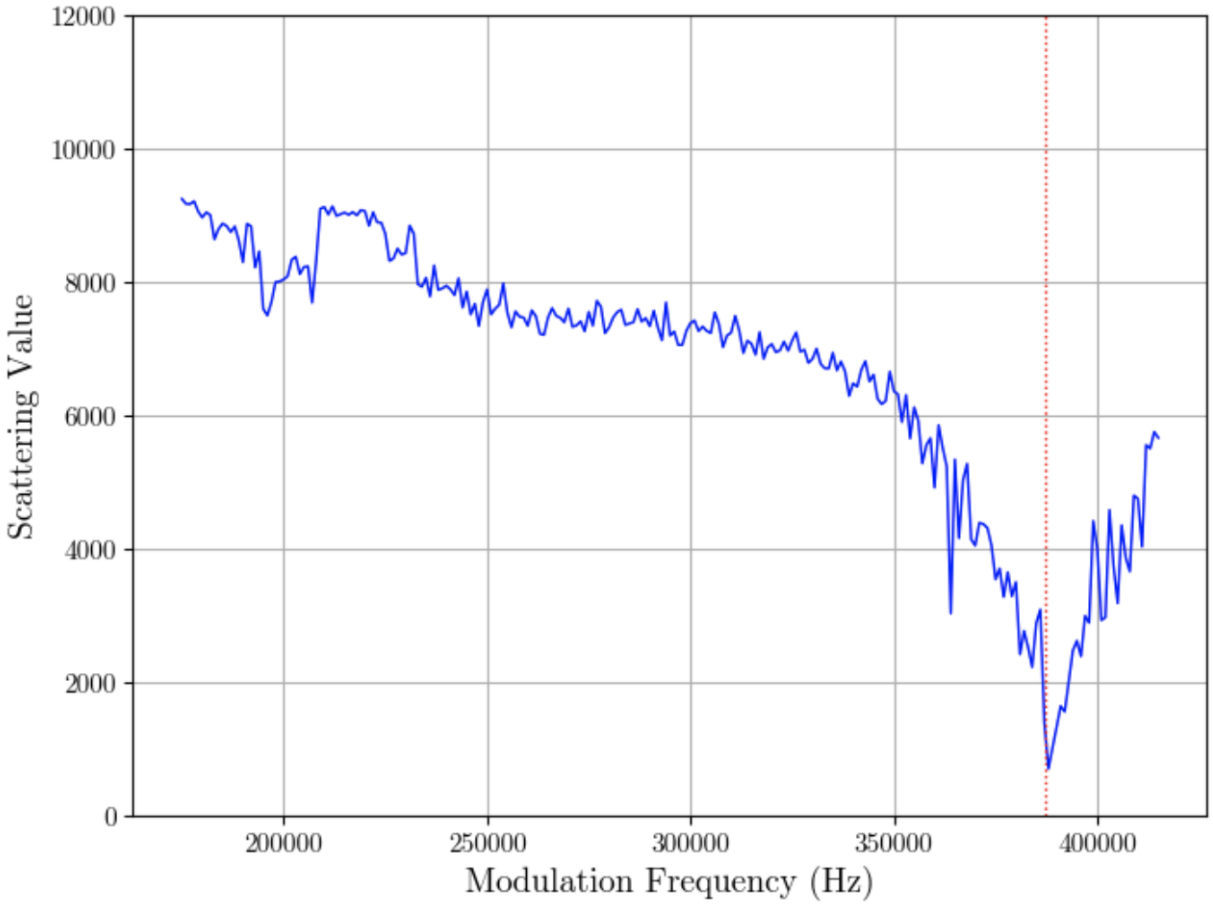


Figure 5.4: Modulation frequency scan for a voltage of 39.5 V and endcap voltage of 2.0 V. The dotted red line indicates the frequency used in the static-frequency simulation run (approximately 387 kHz). While a subtle dip in scattering is observed near the analytically calculated resonance at $\approx$ 194.8 kHz, the dominant resonance appears at twice that frequency, suggesting a stronger response at the second harmonic.

One plausible explanation for this behavior is that the modulation field is coupling more efficiently to a nonlinear response mode of the system — in other words, a second harmonic of the fundamental secular motion. In nonlinear driven oscillatory systems, it is somewhat established that energy can be transferred into overtones or subharmonics when the system is driven at integer multiples (or fractions) of its base frequency [36]. Although the pseudopotential approximation treats the ions as harmonic oscillators near equilibrium, the actual Coulomb crystal is a coupled, many-body system with anharmonic corrections that may enable strong responses at higher-order harmonics [37]. This would mean the Be^+ fluo-

rescence suppression we observe does not strictly correspond to motion at the fundamental secular frequency, but rather to one of its harmonics where the overall motional amplitude or heating is more pronounced.

Another factor to consider is the role of modulation amplitude. At higher modulation amplitudes — like those used in this calibration run — ions may enter a nonlinear regime where energy is no longer confined to the primary secular frequency. Instead, resonant coupling can occur across a broader spectrum, particularly near higher-order multiples where the effective energy transfer is maximized [36]. It's also possible that motion at twice the secular frequency (which may appear more dynamically symmetric) leads to more consistent disruptions in Be^+ cooling, thereby producing a more visible drop in scattering signal. This could be especially true if Be^+–O^+ coupling is stronger in that regime, which would translate the otherwise subtle motion of O^+ into a measurable signal with enhanced fidelity. The structure of this nonlinear behavior aligns with theoretical predictions from solutions to the Mathieu–Hill equation, which describe the complex, parameter-dependent dynamics of ions in RF traps [38].

This observed frequency-doubling behavior raises a practical question for experimental design: should modulation frequency be tuned to the theoretical secular frequency, or instead to the empirically observed scattering minimum? While a more detailed investigation is warranted to fully resolve the mechanism,[4] we provisionally proceed by tuning to the second multiple of the calculated secular frequency — the choice that aligns with the dominant dip observed in Figure 5.4. This assumption is later validated by the overall behavior of the system across a wide range of parameter configurations.

Given this evidence, we adjust our strategy and rerun simulations using the empirically observed resonance frequency. We thus obtain the following heatmap shown in Figure 5.5.

This figure reflects a refined sweep at a modulation frequency of approximately 390 kHz, corresponding to the second harmonic of the corrected radial secular frequency discussed

[4]This remains an open question and falls outside the primary scope of this thesis, though future work could investigate it in depth.

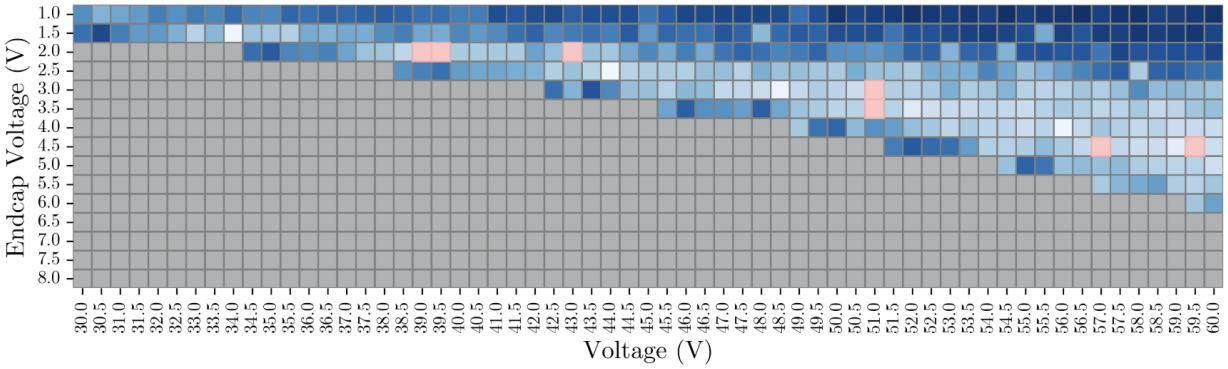
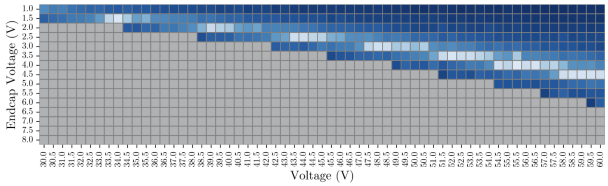


Figure 5.5: Updated heatmap showing scattering behavior. Unlike the previous version, this plot dynamically computes the modulation frequency for each parameter set, using twice the theoretical values derived from Chapter 1 — consistent with observed resonance behavior. Configurations where trap stability breaks down are shown in gray, while those with `NaN` values are indicated in pink. The blue color gradient matches that of Figure 5.3. The modulation amplitude used here is 1.0, to be compared with Figure 5.6.

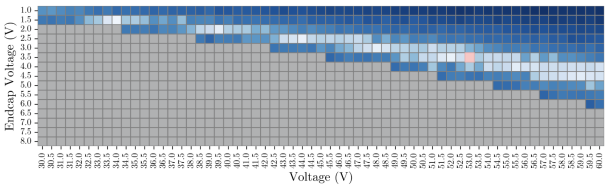
earlier. The white regions — indicative of low scattering — now appear much more sharply defined and consistent with what we expect based on our empirical observations. This provides further support for the idea that our system responds most strongly to modulation near twice the nominal secular frequency, possibly due to nonlinearities or second-order motional coupling. The clearest low-scattering "ridge" still occupies a narrow band in the voltage-endcap voltage space, providing us with a target subspace for high-resolution dissociation detection.

Having established the importance of resonant frequency, we next explore how the drive amplitude modulates the response landscape. To further probe the dynamics of this response, we vary the modulation amplitude across four distinct values, holding all other parameters fixed. These plots reveal how scattering behavior changes as the drive strength is increased — particularly how the system transitions from clean, well-defined resonance dips to more chaotic or smeared-out scattering patterns. The progression is shown below:

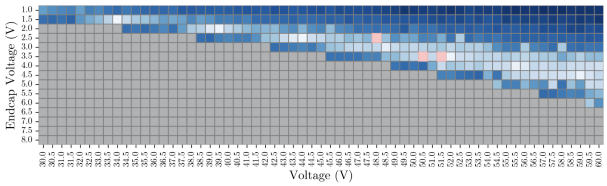
These results illustrate a clear transition from coherent to chaotic behavior as drive strength increases. At low modulation amplitudes, the system's response is highly selective. The crystal remains well-ordered, and only configurations tuned precisely to the correct



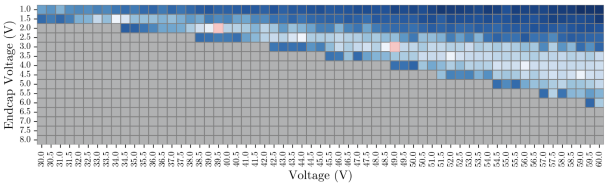
(a) Modulation Amplitude = 0.2



(b) Modulation Amplitude = 0.4



(c) Modulation Amplitude = 0.6



(d) Modulation Amplitude = 0.8

Figure 5.6: Scattering heatmaps for a range of modulation amplitudes. As the modulation amplitude increases, the structure of the scattering signal becomes progressively more fragmented and irregular. At lower amplitudes, the signal exhibits well-defined, coherent streaks that are easier to interpret and trace across parameter space. In contrast, higher modulation amplitudes introduce greater variability and noise, making it more difficult to identify consistent trends or regions of optimal performance. The blue color scale used here is identical to that used in both prior heatmap figures.

secular resonance yield a drop in scattering, making these regions particularly useful for calibration. As the amplitude increases, however, the precision of this resonance erodes. The regions of suppressed scattering widen, flatten, or in some cases begin to show signs of secondary minima — all of which make interpretation more difficult.

This behavior is expected in driven nonlinear systems, where large modulation amplitudes can push ions into regimes of complex dynamics, including parametric instabilities, off-resonant excitation, or unwanted heating. While stronger drives can increase signal amplitude, they also increase the likelihood of false positives — configurations that appear to produce signal dips but do so for reasons unrelated to true resonance.

Equipped with this refined picture, we are ready to scale our analysis to higher dissociation states. Now that we have a concrete mapping of the parameter space associated with low scattering — at least for the first dissociation step — we can use this as a foundation. The next challenge is to apply similar analysis to higher-order dissociation steps, where the

presence of multiple O^+ ions introduces more subtlety into the motional dynamics and their detectability via Be^+ fluorescence.

5.2.2 Number Resolution Step

We now shift focus from the first dissociation event to the more complex problem of resolving multiple ions in the trap. We begin by simulating a total of 84,300 unique parameter configurations, each representing a distinct combination of voltage, endcap voltage, modulation amplitude, and dissociation level. These results are compiled into a structured dataframe for downstream analysis. During preprocessing, we exclude all rows containing `NaN` values. While future work may investigate the source and structure of these missing entries in more detail — potentially enabling principled imputation or uncertainty modeling — we opt here for simple removal. The fraction of missing data is small,[5] and its exclusion does not meaningfully impact the size, balance, or predictive capacity of the dataset.

To enable fair comparison across parameter configurations, we adopt a unified cost metric rooted in the sharpness and coherence of each trace. We evaluate signal quality using the same composite cost function defined previously in Section 4.3: a weighted combination of normalized slope and inverse R^2, which captures both the sharpness and coherence of the dissociation trace. Specifically, the cost is calculated as $\text{Cost} = 0.8 \cdot \texttt{Slope\_norm} + 0.2 \cdot \texttt{R-sq\_norm}$, where lower values indicate stronger and more consistent signals. This scoring function provides a standardized basis for comparing configurations across the full simulation dataset.

This weighting reflects the intuition that signal sharpness is more indicative of physical detectability than noise correlation alone. Our choice thus balances the sharpness and reliability of the dissociation signal while providing a meaningful scalar target for classification and optimization tasks throughout the modeling pipeline. The decision to weight the slope more heavily stems from its greater direct relevance to dissociation detection: a steep slope

[5] Approximately 8.9% of rows contain missing values. While not negligible, this fraction is unlikely to bias results significantly, though more rigorous treatment is a worthwhile direction for future work.

reflects the physical transition we aim to resolve, whereas a high R^2 simply ensures that the trend is not noisy or erratic.

It is important to emphasize that this is a preliminary cost function — simple and interpretable, but not exhaustive. We will introduce more nuanced formulations in the Appendix for future work that incorporate uncertainty quantification, multi-stage behavior, and KNN-based stability scores. These richer cost functions aim to better capture the complexity of dissociation behavior in the presence of noise and variability. For the current modeling stage, however, this 80/20 formulation offers a transparent, intuitive starting point that aligns well with our physical expectations and provides a clear baseline for this study.

Table 5.1 summarizes a sample of high-performing configurations under this cost metric. Each entry corresponds to a full simulation run, including its control parameters (voltage, endcap voltage, and modulation amplitude), computed slope, R^2 value, and resulting cost function score. Note that all simulations listed here use a modulation amplitude of 0.4, indicating an interesting trend and isolating the influence of voltage and endcap parameters.

Table 5.1: Sample configurations and corresponding cost function results for the parameters with the lowest cost functions across the parameter set. Lower cost indicates better experimental quality.

Voltage (V)	Endcap Voltage (V)	Mod Amp	Slope	R^2	Cost Function
48.2	3.5	0.4	-506.53	0.880	4.77
55.4	5.0	0.4	-544.90	0.753	4.93
48.0	3.5	0.4	-482.82	0.914	5.58
58.7	5.5	0.4	-486.03	0.862	6.41
59.0	5.5	0.4	-499.44	0.788	7.06
50.1	4.0	0.4	-466.93	0.876	7.32
55.1	5.0	0.4	-488.41	0.794	7.62
52.0	4.5	0.4	-473.91	0.823	7.94
40.4	2.5	0.4	-433.89	0.925	8.39
53.4	4.5	0.4	-478.23	0.771	8.72

Beyond cost alone, we also highlight a more stringent set of traces: those in which the scattering signal exhibits a strictly decreasing trend across all dissociation steps. That is, each jump from a particular dissociation level to the following is negative. These parameter

sets encapsulate the exact behavior we set out to find. In the context of number resolution, strictly decreasing scattering values signify that each additional O^+ ion entering the trap contributes a clear, measurable change in the system's response.[6] No reversals, no plateaus — just clean, monotonic decline. This type of behavior is a crucial standard for resolving individual ion dissociation events, and such configurations are exceedingly rare and represent the ideal case for number-resolved detection: they demonstrate that the system is operating in a regime where each incremental ion alters the dynamics in a quantifiable and consistent way. Table 5.2 highlights the 11 such configurations uncovered through our analysis alongside the extracted slope and R^2 value from a linear fit to each trace.

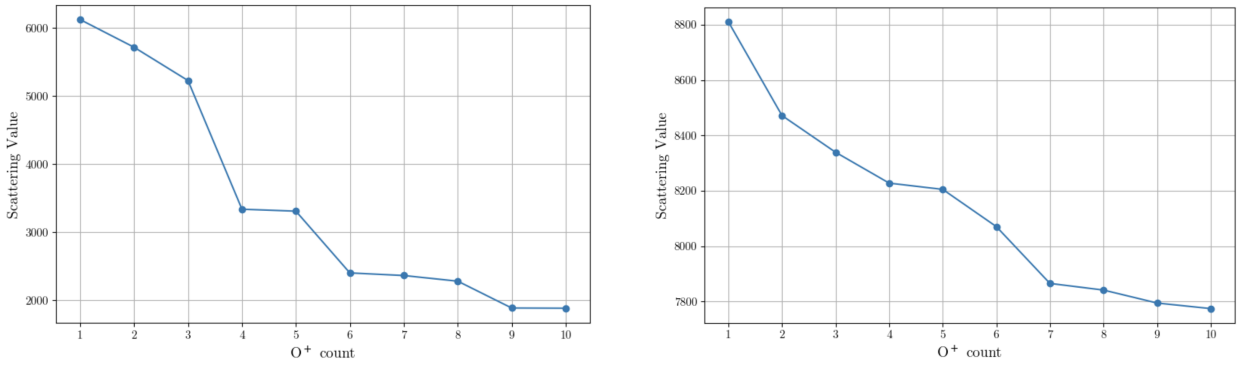
Table 5.2: Configurations with strictly decreasing scattering traces across all dissociation steps.

Voltage (V)	Endcap (V)	Mod Amp	Slope	Intercept	R^2
56.6	2.0	0.2	-116.37	8868.47	0.922
51.7	1.0	0.2	-86.29	9134.32	0.928
53.5	1.0	0.2	-71.26	9113.31	0.793
48.2	3.5	0.4	-506.53	6234.83	0.880
57.6	2.0	0.2	-108.93	8828.00	0.952
40.7	1.0	0.2	-119.87	8579.15	0.883
57.2	1.5	0.2	-101.48	9127.82	0.924
56.4	1.0	0.2	-77.78	9237.91	0.939
55.4	2.0	0.2	-107.74	8732.88	0.926
56.5	5.0	0.6	-309.16	4353.56	0.756
40.9	2.5	0.4	-356.28	5958.13	0.935

These results are strong candidates for benchmarking dissociation detection, as they show not only clean monotonic decline in scattering signal, but also favorable linear fits with steep slopes and high R^2 values. Such traces may be used later to validate the machine learning model's classification fidelity, its capacity to interpolate over regions of clearly structured physical behavior, and its robustness under random perturbation.

Notice in Figure 5.7 that even simulations with strictly decreasing scattering values often

[6]Strictly decreasing here refers only to the mathematical condition that each successive value is lower than the one before. While this ensures directional consistency, the magnitude of change may still be small.



(a) (b)

Figure 5.7: Two strictly decreasing trends extracted from the simulation dataset. (a) highlights the parameter set with the steepest (most negative) slope, while (b) highlights the parameter set with the largest minimum drop across dissociation drops.

exhibit poor resolution at Jump 4 to 5. To better understand how the scattering signal evolves across the dissociation sequence, we compute the average change in scattering between successive states across the full dataset. Table 5.3 shows these average "jumps," which provide a clear picture of how sharply the signal tends to decay as additional O^+ ions are introduced.

Table 5.3: Average change in scattering signal between consecutive dissociation steps.

Dissociation Step	Average Scattering Change
$1 \to 2$	-342.15
$2 \to 3$	-169.88
$3 \to 4$	-78.72
$4 \to 5$	-32.34
$5 \to 6$	-14.73
$6 \to 7$	-14.01
$7 \to 8$	$+9.68$
$8 \to 9$	-6.22
$9 \to 10$	-23.19

The results show that early dissociation steps, particularly Jumps 1 to 2 through 3 to 4, tend to have large and consistent signal drops. After Jump 4 to 5, however, the average decline in scattering diminishes substantially, and by Jump 7 to 8, we even observe a *positive* mean change — a counterintuitive result suggesting local non-monotonicity. These trends

indicate that a simple linear fit may be inadequate for capturing the behavior of the full dissociation trace. While early regions are steep and regular, later steps exhibit flatter, more erratic behavior that is increasingly sensitive to stochastic effects or small perturbations in the system.

To further quantify this behavior, we count how many simulations exhibit drops of at least 100 units between each pair of dissociation steps. Table 5.4 reports both the raw counts and percentages, offering a complementary perspective on how frequently large, resolvable transitions occur.

Table 5.4: Number and percentage of simulations with at least a 100-unit drop between dissociation steps.

Dissociation Step	Count of Decreasing Rows	Percentage (%)
$1 \rightarrow 2$	4997	67.65%
$2 \rightarrow 3$	4017	54.38%
$3 \rightarrow 4$	3537	47.88%
$4 \rightarrow 5$	3259	44.12%
$5 \rightarrow 6$	3054	41.34%
$6 \rightarrow 7$	3081	41.71%
$7 \rightarrow 8$	2827	38.27%
$8 \rightarrow 9$	2933	39.70%
$9 \rightarrow 10$	2848	38.55%

Together, these tables reveal a structural transition: the dissociation signal begins with large, easily resolved drops but gradually flattens out. Later steps show smaller average declines and less consistent behavior, especially around Jump 7 to 8, where the signal sometimes increases. This suggests that the dissociation process is not uniformly linear — a pattern likely rooted in physical changes in the crystal structure or ion dynamics as more O^+ ions are added. Although we do not make explicit attempts to fit non-linear models to this trend, this may be a valuable search in future work. To better visualize how this trend evolves across the dissociation sequence, we turn to a threshold-based analysis of scattering drops.

Figure 5.8 further illustrates this trend by showing the percentage of simulations that

meet several drop thresholds (100, 200, 300, etc.) across the dissociation steps. As the sequence progresses, the likelihood of observing large, clean drops falls off significantly.

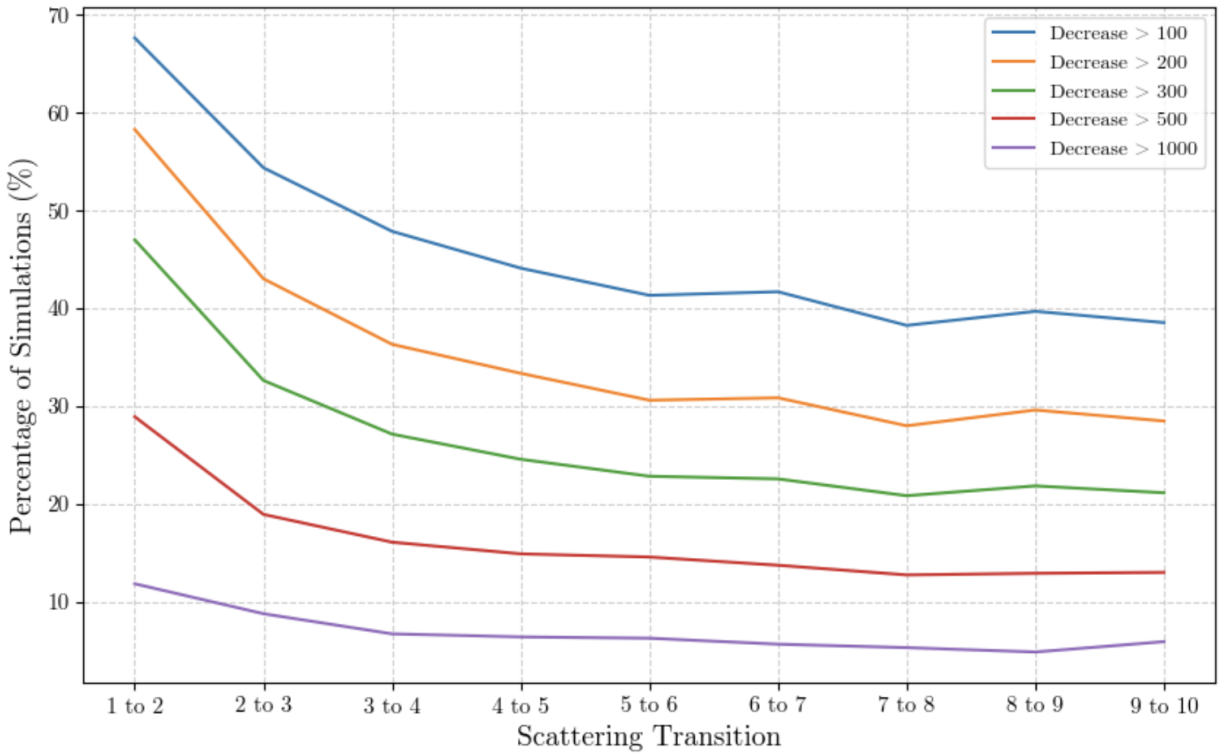


Figure 5.8: Percentage of simulations that show a scattering decrease of at least 100, 200, 300, 500, or 1000 between consecutive dissociation levels. Large, resolvable drops occur most frequently in the early steps, while later transitions become increasingly ambiguous.

Importantly, this diminishing signal resolution has direct consequences for how we interpret and model later dissociation steps. Namely, the diminishing resolution in later steps weakens the reliability of dissociation-state classification and suggests the need for more flexible modeling. Piecewise or nonlinear regression strategies — explored in later sections — may help capture these transitions more effectively, particularly where standard linear fits break down. By identifying which steps are most prone to ambiguity, we can tailor both our physical models and classification strategies to focus predictive power where it matters most.

To move beyond step-by-step dissociation behavior, it is useful to zoom out and examine how control parameters shape experimental performance as a whole.

We now turn our attention to the broader structure of the parameter space and its relationship to the cost function. To better understand how parameter choices relate to overall performance, we plot the cost function against applied RF voltage across all simulations. As shown in Figure 5.9, there is no clean trend or monotonic behavior; instead, low-cost regions are scattered nonlinearly throughout the voltage range. Promising configurations do not cluster around any single voltage value, suggesting that good performance arises from more subtle combinations of control parameters rather than simple tuning of a single variable.

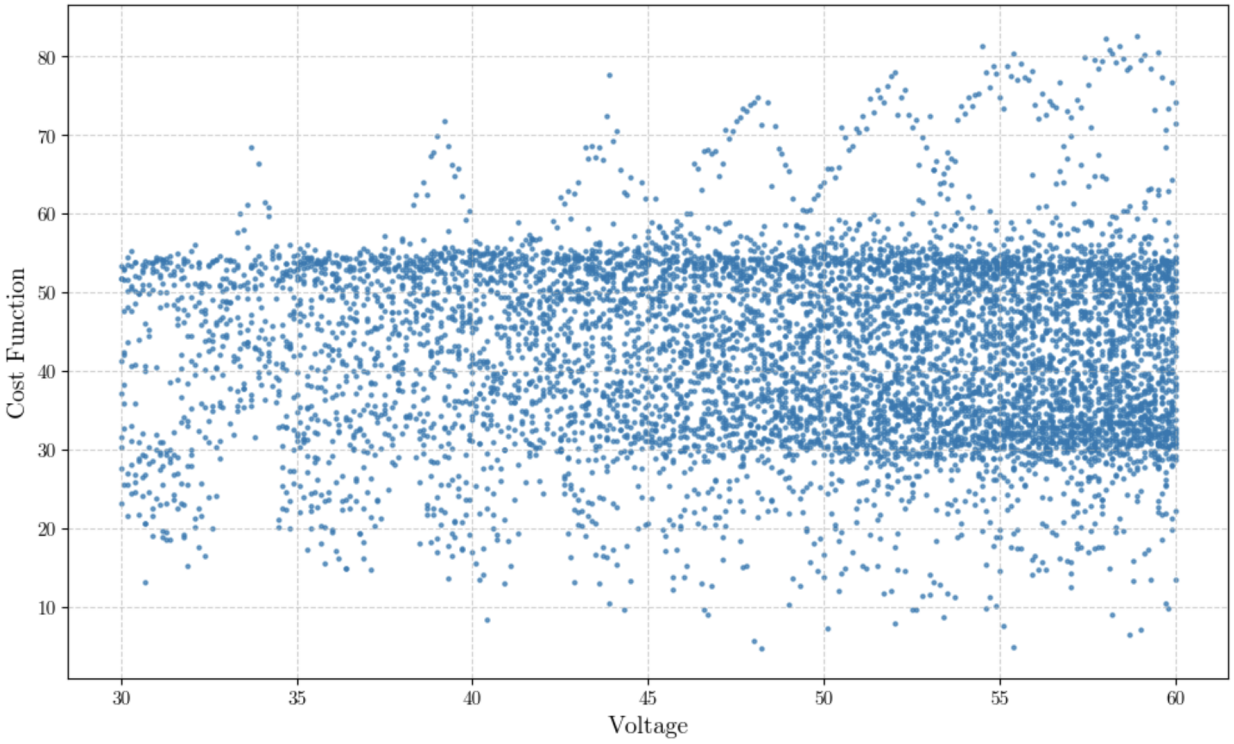
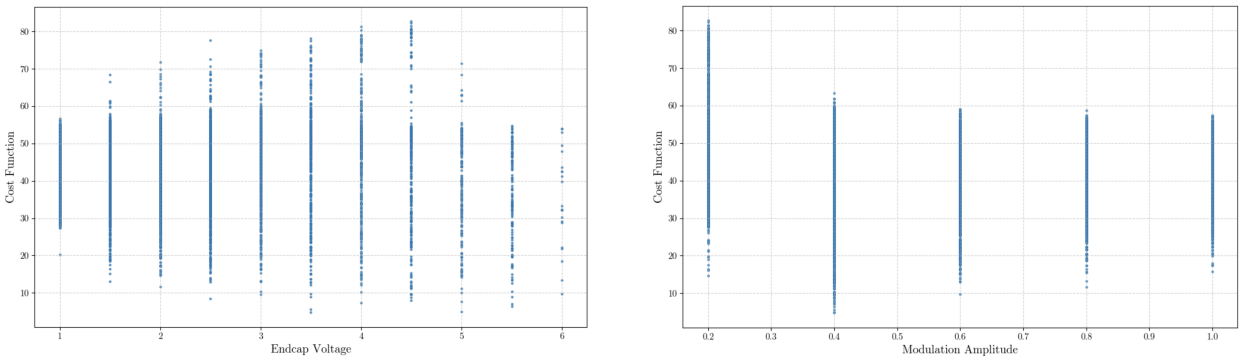


Figure 5.9: Scatter plot showing the distribution of cost function values as a function of applied RF voltage. While some clustering is apparent, the overall cost landscape is highly nonlinear, indicating that optimal configurations result from subtle interactions between multiple parameters rather than simple tuning of voltage alone. The structure of the plot includes a dense horizontal streak — where the majority of simulations reside — along with a faint sinusoidal pattern that weaves around it, hinting at periodic structure or coupled parameter dynamics within the configuration space.

This complex cost landscape reflects interactions between RF voltage, endcap voltage, and modulation amplitude, and points to the need for multidimensional models that can

learn these joint dependencies. Decision trees, random forests, and other ensemble methods are particularly well suited for this task. In fact, the very lack of simple trends or isolated optimal regions in parameter space provides foundational evidence that justifies our modeling strategy: machine learning models that can capture nonlinear relationships, adapt to sparse local patterns, and integrate interacting features are not just helpful, but essential. The structure of the parameter space itself validates the choice of flexible, data-driven approaches like those we explore.

Finally, we plot the cost function against endcap voltage in Figure 5.10a. While the dependence is more scattered than in the case of RF voltage, we still observe clustering of low-cost configurations around specific voltage ranges. These may correspond to regions of optimal axial confinement, which affect crystal geometry and motional coupling.



(a) (b)

Figure 5.10: Cost function behavior across two parameters. Left: Endcap voltage exhibits mild influence on configuration quality. Right: Modulation amplitude shows more pronounced effects, particularly at low amplitudes.

This approach is extended to additional control variables to assess their individual influence. We perform a similar analysis for modulation amplitude, shown in Figure 5.10b. Unlike the other parameters, modulation amplitude appears to exert a more complex, nonlinear effect on the cost landscape. Some local minima occur at intermediate values — likely

reflecting trade-offs between signal strength and crystal stability during modulation.[7]

These parameter-specific trends highlight the deterministic structure of the cost landscape as seen from single-run simulations. However, real experimental systems — and by extension, high-fidelity simulations like QLICS — exhibit inherent variability that cannot be captured by one trial alone. To address this, we now turn to a Monte Carlo framework that systematically quantifies uncertainty across repeated runs of identical configurations.

5.2.3 Monte Carlo Analysis

Up until this point in our results discussion, simulations have been treated as deterministic — that is, each configuration has been assumed to produce a fixed scattering outcome. However, as described in Section 2.4, QLICS introduces stochasticity at multiple stages, both internally and through its coupling with LAMMPS. These sources of randomness include thermal noise, probabilistic photon scattering, and numerical variation in ion trajectories. As a result, a single simulation run does not fully characterize a configuration. To capture the inherent variability, we instead rerun the same configuration file multiple times to generate an empirical distribution of outcomes. This approach allows us to estimate statistical properties — such as mean scattering, variance, or confidence intervals — and to build models that are robust to noise rather than overfitted to a single realization.

This variability provides the foundation for constructing a Monte Carlo model, in which each configuration is treated as a distribution rather than a single-point estimate. By repeatedly sampling from this distribution — through re-running the same configuration multiple times with injected noise — we can simulate the probabilistic behavior of the system and propagate uncertainty through the optimization and classification pipelines. This allows us to evaluate not only the mean performance of a configuration, but also its stability and

[7]The reader may notice that voltage was sampled more densely than either modulation amplitude or endcap voltage. This decision emerged for two main reasons: (i) voltage consistently ranked as the most important feature in early models trained on evenly sampled, standardized data; and (ii) interpolation models struggled more with voltage than with other parameters, suggesting that voltage-related variations induce greater local randomness in system behavior. However, future work should direct greater attention to these other features.

reliability across repeated runs.

Importantly, we make no assumptions about the shape of the underlying distribution for each configuration's output. While some modeling approaches might default to Gaussian or uniform priors, and QLICS itself introduces approximations for photon scattering and randomized uniform distributions for initial ion positions, we deliberately avoid imposing any particular parametric form on the observed scattering data. This decision reflects the complex, layered nature of the stochasticity present in the system: initial ion positions are drawn uniformly within a spherical volume, photon counts are sampled as discussed in Chapter 2, and modulation responses emerge from a mixture of these random elements under resonance-sensitive dynamics. In this context, assuming a fixed distribution — even a physically motivated one — risks mischaracterizing the true variability, especially in regimes where the combined effects are nonlinear or system state–dependent. Instead, we treat each simulation as a draw from an unknown, potentially nonparametric distribution, and construct empirical models that capture this behavior directly. By doing so, we preserve the structure and irregularities of the simulated outcomes and avoid distorting the data to fit a simplified analytical mold.

Statistic	**Count**	**Mean**	**Median**	**Std. Dev.**	**Min**	**Max**
Scattering Values	120	7775.12	7778.66	159.03	7398.23	8162.45

Table 5.5: Summary statistics for photon scattering values across multiple runs for a unique parameter set. Random behavior was assessed based on local multiprocessing rather than utilizing random seeds via HPC batch run.

Specifically, we generate 10,000 bootstrap[8] resamples (each of size 120) from the empirical distribution of scattering values. For each resample, we calculate the proportion of values that fall within symmetric intervals around the sample mean (e.g., within ± 50, ± 100, ..., ± 400). Averaging across bootstrap samples produces an empirical estimate of how frequently scattering values fall within these ranges for typical simulations, while the variation across

[8]A bootstrap resample is created by randomly drawing (with replacement) from the original sample of 120 scattering values, preserving sample size while allowing duplicates.

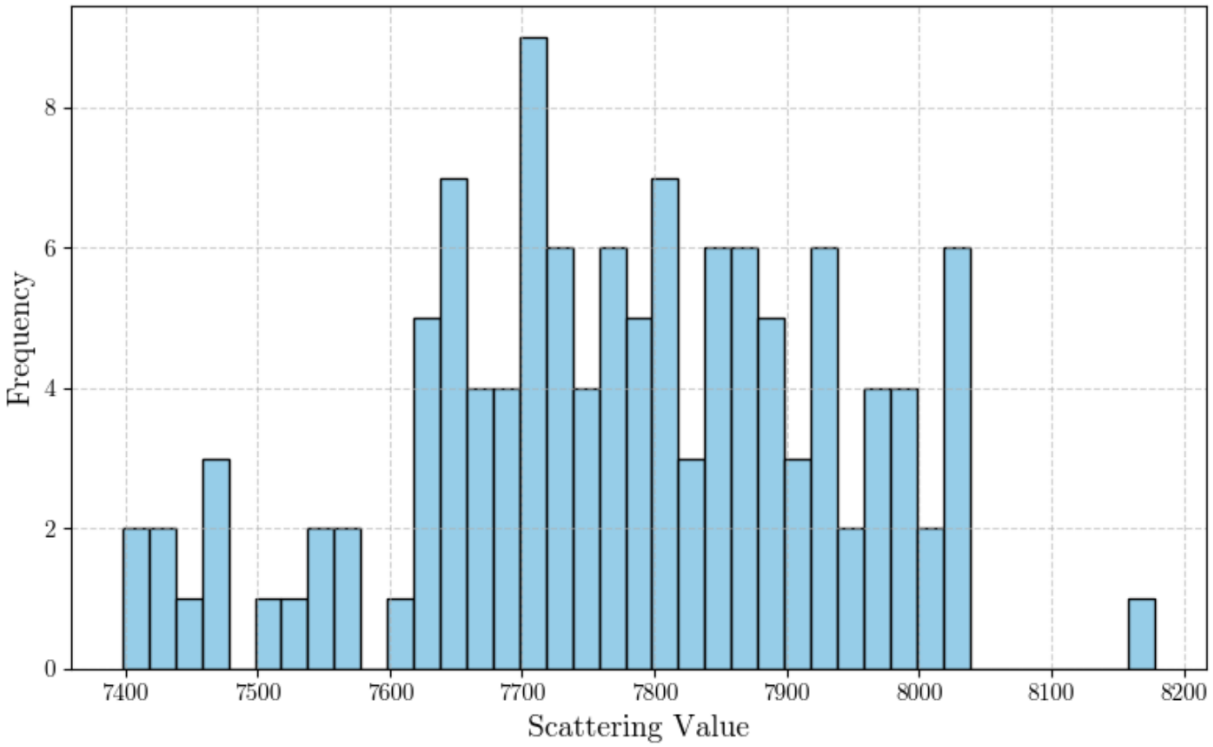


Figure 5.11: Histogram showing the distribution of scattering values for a single dissociation (one atomic oxygen in the trap), with bin size 20. Note that this distribution results from complex interactions between underlying distributions. We assume throughout this thesis that random behavior, particularly in the spread around the mean at a specific scattering value, is independent of specific parameters. Random sampling and probabilistic estimates confirm the basis for this assumption.

Range	**Mean Proportion**	**Standard Deviation**
±50	0.2372	0.0411
±100	0.4717	0.0482
±150	0.6641	0.0441
±200	0.7787	0.0419
±250	0.8881	0.0394
±300	0.9363	0.0211
±350	0.9681	0.0173
±400	0.9984	0.0045

Table 5.6: Bootstrapped proximity proportions: average probability that a value falls within a given range of the sample mean across 10,000 bootstraps.

bootstraps provided a measure of uncertainty in those estimates.

These proximity probabilities describe how tightly values tend to cluster around their

sample mean in repeated simulation contexts and can be used to inform expectations about how a given scattering value might vary across repeated trials. For example, if our analysis shows that 90% of values in typical simulations fall within ±250 of their respective means, then we can say the following: if a given scattering value x is generated from a similar simulation setup, and if the underlying distribution behaves comparably to those observed,[9] then there is approximately a 90% chance that a second simulation under identical conditions would yield a result within $[x - 250, x + 250]$.

It is important to note that this does *not* imply a 90% confidence interval for the mean centered on x; such an inference would require additional assumptions about symmetry or parametric form. Instead, our interpretation is predictive in nature: the proximity intervals reflect empirical uncertainty due to simulation randomness, not epistemic uncertainty about a population parameter. In this way, these intervals provide a principled basis for constructing Monte Carlo-derived error bars around individual scattering outcomes, grounded in the empirical behavior of the simulation system itself.

To rigorously apply this framework, we now build a Monte Carlo model that leverages the empirically derived proximity intervals to simulate variability in dissociation traces. Rather than assuming a predefined distributional form, we use the bootstrapped proximity probabilities from Table 5.6 to construct nonparametric uncertainty bands around each scattering value. Specifically, we interpret each proximity band — such as ±250 or ±300 — as a cumulative probability threshold and sample uniformly within the region bounded by two adjacent bands. For instance, if the probability of a scattering value falling within ±300 of the mean is 93.63% and the probability within ±250 is 88.81%, then we estimate that approximately 4.82% of outcomes lie in the range between 250 and 300. A random draw falling into this band would then be uniformly sampled within the interval $[x-300, x-250] \cup [x+250, x+300]$,

[9] This is a significant assumption; while random sampling supports its validity in practice, a dedicated study would be needed to fully verify this behavior.

where x is the original scattering value.[10]

We repeat this process independently for each dissociation level in a trace. Each value is perturbed according to its corresponding empirical uncertainty profile, thereby generating synthetic scattering traces that reflect the stochastic behavior of the physical system without assuming Gaussianity or symmetry. This method treats the bootstrapped proximity proportions as a lookup table for uncertainty modeling: the in-between regions serve as quantized bands with uniform sampling applied to approximate local variability. When repeated across many trials, this Monte Carlo procedure produces an ensemble of plausible dissociation traces, capturing both the central tendencies and the long-tail behavior of experimental outcomes. These synthetic traces can then be passed through the same cost function and classification pipelines described previously, enabling robust downstream inference that is sensitive to realistic uncertainty in the input data.

Suppose we want to evaluate the probability that out strictly decreasing parameter sets stay strictly decreasing. Now equipped with this Monte Carlo approach, such analysis is simple. Monte Carlo analysis[11] reveals that the 11 rows exhibiting strictly decreasing behavior under ideal conditions are exceptionally sensitive to noise. When subjected to 1000 randomized perturbation trials, each designed to reflect empirically observed variability in the scattering signal, the probability that these configurations retain their strictly decreasing profile is strikingly low. In nearly all cases, the estimated probability of maintaining this structure is below 1%, with most falling near or exactly at 0%. The highest observed probability is only 8.1%, underscoring the fragility of this behavior under noise injection. Given that strict monotonicity requires all ten successive values to decrease under random fluctuation, these results emphasize how unlikely it is for such patterns to persist without significant

[10]Uniform sampling within each band is justified because the bootstrap-derived probabilities specify only the proportion of values falling between adjacent proximity intervals—not their distribution within. In the absence of further structure, a uniform assumption ensures that we do not introduce artificial skew or density that isn't empirically supported.

[11]Each strictly decreasing configuration was perturbed 1000 times using noise sampled from empirically derived proximity bands. After each trial, we checked whether the perturbed trace remained strictly decreasing across all dissociation steps. The final probability reflects the fraction of trials in which strict monotonicity was preserved.

stabilization. This suggests that apparent strict monotonicity may often reflect numerical coincidence or transient behavior, rather than robust or reproducible system dynamics.

The Monte Carlo framework provides a principled method for quantifying variability in dissociation traces. By running repeated stochastic perturbations on each configuration, we can compute predictive uncertainty intervals — such as the 5th to 95th percentile — that reflect the expected spread of the signal due to inherent system randomness. These intervals offer a more realistic view of each configuration's stability, helping to distinguish between parameter sets that are merely sharp on average versus those that consistently exhibit steep, monotonic behavior.

Beyond global uncertainty estimates, the model enables targeted probabilistic queries. For instance, we can evaluate the likelihood that a critical jump (e.g., from step 5 to 6) exceeds a threshold magnitude. High probabilities suggest reliable state resolution, while low or inconsistent values signal fragility or susceptibility to failure — valuable insights when selecting robust configurations for downstream use.

In addition to evaluation, the Monte Carlo approach may also be utilized to expand the training dataset. By generating multiple noisy versions of each configuration, we create examples that reflect the kinds of variation that might occur in real experiments. This helps prevent overfitting and improves generalization, especially when measurements are affected by noise. In this way, the Monte Carlo model contributes to both robustness analysis and the development of models that perform reliably under realistic conditions. Beyond improving model generalization, the Monte Carlo approach offers additional insight into the expected variability of dissociation outcomes. By characterizing this variation explicitly, we can begin to define what constitutes a "realistic" performance band for any given configuration — and, in turn, make more informed optimization decisions. The next section formalizes this idea by framing our evaluation in terms of a mixed strategy equilibrium, where robustness and variability are considered jointly.

5.2.4 A Mixed Strategy Equilibrium

As demonstrated by our Monte Carlo model and corroborated by repeated stochastic simulations, fluctuations of up to ±250 photon counts across dissociation levels are entirely within reason. These variations arise not from noise or numerical instability, but from real, inherent randomness encoded in the simulation architecture. In fact, the synthetic randomness introduced at the beginning of each simulated run likely results in more pronounced variability than what might occur in a physical lab setup. In our framework, each trial begins by placing a fixed number of molecular oxygen ions randomly within the trap — a process that effectively reshuffles the entire configuration from scratch. By contrast, in an actual experiment, molecular oxygen is stochastically loaded once, and its subsequent dissociation proceeds without further explicit spatial redistribution. This distinction leads to greater stochastic dispersion in the simulation and underscores that our error bars represent a kind of upper bound — a conservative estimate of how significantly dissociation signals might vary under uncontrolled conditions.

Despite our efforts to identify a single experimental configuration that yields clean, reliable scattering behavior across all dissociation transitions, the data suggests a fundamental limitation: there is no single state in parameter space that robustly supports steep scattering drops across both early and late dissociation jumps. To be clear, we did succeed in identifying configurations that show strictly decreasing scattering behavior and exhibit high R^2 values — an indication of solid number resolution performance, yet even these configurations break down under random noise injection. Essentially, while the trend is monotonic and fits well, the absolute change in signal is sometimes too narrow to offer reliable resolution in the presence of noise, particularly on narrow jumps (such as four dissociation to five, etc).

What we now seek is a stronger criterion: parameter sets in which scattering jumps exceed a threshold of at least 400. This cutoff provides a form of robustness — a buffer against the natural stochasticity observed in simulation and experiment alike. Even in worst-case scenarios where stochastic fluctuations push a jump slightly downward, configurations with

large signal deltas are far more likely to preserve correct number discrimination. Although it is impossible to eliminate uncertainty entirely, this threshold serves as a practical safeguard, ensuring that even in the presence of variability, the dissociation events remain distinguishable.

To make this concrete, we examine the dissociation transitions in two separate regimes. First, we search for configurations that exhibit consistently large scattering decreases across *early* dissociation steps — from the first ion dissociating through the sixth (*i.e.*, Jump 1 to 2 through Jump 5 to 6). Even under a strict criterion of ≤ -400 drop at each step, we identify two viable configurations, listed below:

Table 5.7: Configurations with ≤ -400 drop across Jumps 1 to 6.

Voltage (V)	Endcap Voltage (V)	Modulation Amplitude	Slope
51.9	4.5	0.4	-440.90
49.7	4.0	0.4	-362.99

In contrast, when analyzing the *late* dissociation steps — Jump 6 to 7 through Jump 9 to 10 — and applying the same ≤ -400 threshold, only a single configuration meets the criterion:

Table 5.8: Configuration with ≤ -400 drop across Jumps 6 to 10.

Voltage (V)	Endcap Voltage (V)	Modulation Amplitude	Slope
54.8	3.0	0.6	-137.03

Crucially, there is *no overlap* between the two sets — the configuration that performs well for late-stage dissociation does not appear in the set that performs well for early-stage behavior, and vice versa. This separation of optimal behaviors across the two regimes indicates an inherent incompatibility — a tension — in tuning a single set of trap and modulation parameters to perform well across the full dissociation range.

This observation motivates a more nuanced solution strategy: rather than attempting to force a one-size-fits-all configuration, we propose a *mixed strategy equilibrium*, inspired by game-theoretic reasoning. In game theory, a mixed strategy involves randomizing over

multiple pure strategies to optimize expected performance. Similarly, in our context, the optimal approach may be to run two distinct experiments in parallel or in rapid succession: one tuned to maximize resolution in the early dissociation regime (steps 1–6), and another optimized for the later transitions (steps 6–10).

The logic is straightforward. Because early dissociation steps are crucial for determining whether dissociation is occurring at all — and because their signal drops are typically the strongest — the first configuration emphasizes maximizing the initial slope. This may involve lower modulation amplitudes and slightly adjusted voltage levels to stabilize the crystal's core structure. The second configuration, meanwhile, can be more aggressive — amplifying the modulation to provoke the final ions into dissociation, even if the signal becomes noisier or the slope shallower. If both configurations are run nearly simultaneously or on alternating cycles,[12] the combined dataset provides high-resolution detection across the full 10-level range, leveraging each configuration where it performs best.

We also analyze the likelihood that selected configurations maintain strictly decreasing scattering behavior under stochastic perturbations. Using our Monte Carlo model with proximity-weighted noise sampling, we simulate each configuration 1,000 times and evaluate whether key subsets of the trace remain strictly decreasing. The first two configurations below are assessed for decreasing behavior from steps 1 through 6, while the third is assessed from steps 6 through 10. All three rows exhibit remarkably high robustness, with probabilities exceeding 94%, suggesting that their strictly decreasing structure is not only present in the original measurement but also highly resilient to simulated noise. This level of consistency distinguishes them from the 11 configurations previously analyzed, where the probability of preserving this structure was nearly zero.

This hybrid protocol does not constitute a workaround — it reflects a deeper structural insight about the system: that parameter regimes favoring early stability are not the same as those that encourage late-stage dissociation. Attempting to force both goals into a single

[12]Likely, sequentially to allow for as stable, consistent conditions as possible.

Table 5.9: Strictly Decreasing Behavior Under Monte Carlo Perturbations

Steps 1–6

Voltage	Endcap Voltage	ModAmp	Prob. Stays Strictly Decreasing
51.9	4.5	0.4	0.990
49.7	4.0	0.4	0.949

Steps 6–10

Voltage	Endcap Voltage	ModAmp	Prob. Stays Strictly Decreasing
54.8	3.0	0.6	0.957

configuration imposes incompatible constraints. By embracing a mixed strategy, we align our experimental design with the underlying dynamics of the system, maximizing performance not through uniformity, but through carefully chosen specialization.

This hybrid strategy, grounded in simulation-derived insight, provides a nuanced framework for defining what success looks like in this system. The natural next step is to ask whether our machine learning model can internalize that framework — distinguishing reliably between promising and poor configurations. To assess this, we begin by evaluating the model's classification performance.

5.3 Machine Learning Model Evaluation

We begin this section by defining a few essential terms and metrics that are used to interpret the performance of classification models throughout the chapter. With these concepts in place, we proceed to evaluate three distinct approaches: a bagged ensemble model using random forests, a boosted ensemble model via gradient boosting, and finally, a local KNN-based interpolation scheme that provides a neighborhood-informed measure of configuration quality. Each method offers a different perspective on the classification task, and together they form a comprehensive framework for assessing both accuracy and stability in model

predictions.

In binary classification, predictions can be categorized as true positives (TP), false positives (FP), true negatives (TN), or false negatives (FN) [20]. A true positive occurs when the model correctly predicts the positive class, while a true negative corresponds to correctly predicting the negative class [19]. A false positive is when the model incorrectly predicts the positive class for a negative instance, and a false negative is when it incorrectly predicts the negative class for a positive instance [19]. These four quantities form the entries of the confusion matrix, which summarizes classification performance [22].

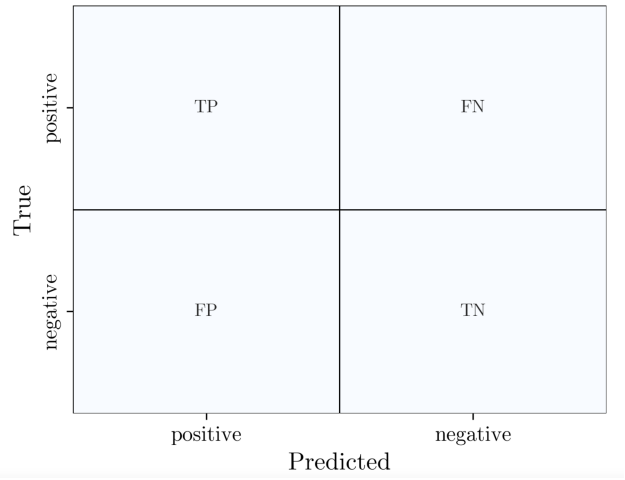


Figure 5.12: A simple confusion matrix showing the placement of true positives, false positives, etc. No need to be confused as the name suggests!

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{5.2}$$

The confusion matrix provides a concise summary of how well our binary classifier performs against the ground truth. Precision tells us how many of the predicted positives were actually correct, reflecting the model's ability to avoid false positives [19]. Recall, on the other hand, measures how many of the true positive instances were successfully identified, indicating how well the model captures actual positives [19]. A high precision means fewer false alarms, while high recall means fewer missed detections. Accuracy, by contrast, simply

refers to the total proportion of correct predictions — both positives and negatives — out of all predictions made.

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.3}$$

The F1 score balances both precision and recall, offering a harmonic mean that penalizes extreme trade-offs [19].

$$\text{Macro F1} = \frac{1}{C} \sum_{i=1}^{C} \text{F1}_i \tag{5.4}$$

Macro F1 averages F1 scores across both classes equally,[13] which is particularly useful in the presence of class imbalance. This metric allows us to evaluate how well the model balances precision and recall across both classes, regardless of imbalance. With this in mind, we now examine the performance of our first model: a bagged ensemble of decision trees trained to classify configurations based on key physical parameters.

5.3.1 Bagging

We begin by evaluating a Random Forest classifier trained to distinguish between "good" and "bad" configurations using three physical input features, with classification determined based on a threshold of the 20th percentile of the cost function defined in Section 5.2.2. The data is split into training and testing sets, and labels are encoded such that "good" corresponds to 1 and "bad" to 0. After training, we compute the relative importance of each feature, measured by how often and effectively it is used in the ensemble's decision trees.

To assess performance, we first train a baseline random forest model using default hyperparameters.[14] The dataset was split into 80% training and 20% testing subsets, and class labels were encoded such that `good` = 1 and `bad` = 0.

[13] In binary classification, this refers to computing the F1 score separately for class 0 and class 1, and then averaging the two.

[14] Default parameters in `RandomForestClassifier` include `criterion='gini'`, `max_depth=None`, `min_samples_split=2`, `min_samples_leaf=1`, `max_features='sqrt'`, and `bootstrap=True`.

Table 5.10: Feature importance scores from the baseline Random Forest classifier.

Feature	Importance (%)
Voltage	64.43
Endcap Voltage	23.29
Modulation Amplitude	12.27

Initial evaluation shows that the trained model achieves a test accuracy of 83.15%, with stronger performance on the majority class (`bad`) and lower recall for the minority class (`good`). Precision and recall imbalances are reflected in the macro-averaged F1 score of 0.74, indicating that while the classifier performs well overall, it initially struggles to consistently identify the minority class. This difficulty is reflected in the minority class's precision (0.61) and recall (0.55), both of which are less than ideal. This suggests that high-quality configurations are harder to distinguish under the current feature set and class imbalance, motivating the need for targeted reweighting and hyperparameter tuning to improve recall.

Table 5.11: Classification report for the initial random forest classifier on the test set.

Class	Precision	Recall	F1 Score	Support
bad	0.88	0.91	0.89	1165
good	0.61	0.55	0.58	313
accuracy			0.83	1478
macro avg	0.75	0.73	0.74	1478
weighted avg	0.83	0.83	0.83	1478

To improve upon this baseline, we next apply hyperparameter optimization using two standard techniques: grid search and randomized search.[15] Both methods use 5-fold cross-validation[16] and macro F1 as the evaluation metric.

The grid search explored 96 parameter combinations across a defined search space and identified the best-performing model with the following hyperparameters: `n_estimators` =

[15]Grid search exhaustively evaluates all combinations of specified hyperparameters, while randomized search samples a fixed number of combinations randomly, offering faster results when the hyperparameter space is large.

[16]In 5-fold cross-validation, the dataset is split into 5 parts; the model is trained on 4 parts and validated on the remaining one, rotating this process across all 5 folds to ensure robust performance estimates.

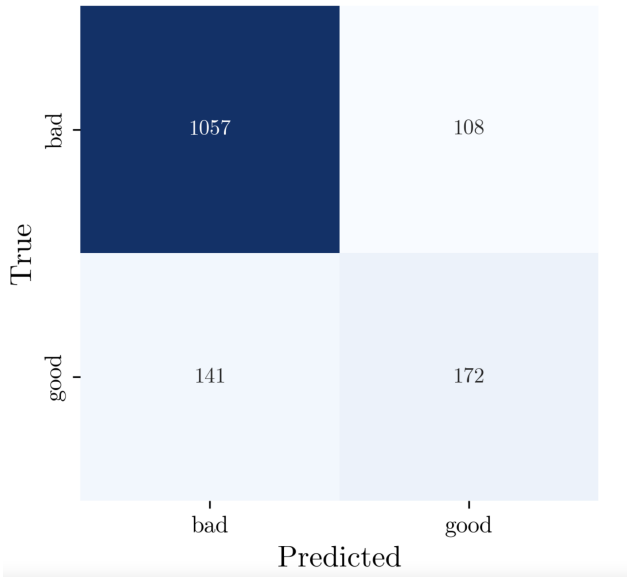


Figure 5.13: Confusion matrix showing performance on the testing set for the initial bagging (Random Forest) model.

300, `max_depth` = 10, `min_samples_split` = 5, `min_samples_leaf` = 2, `max_features` = `"sqrt"`, and `class_weight` = `"balanced"`. This model achieved a macro F1 score of 0.8158.

To further probe the hyperparameter space, we then apply randomized search over 400 sampled configurations. The best model from this approach slightly outperformed the grid search, achieving a macro F1 score of 0.8175. The optimal parameters were: `n_estimators` = 405, `max_depth` = 10, `min_samples_split` = 3, `min_samples_leaf` = 3, `max_features` = `"log2"`, and `class_weight` = `"balanced_subsample"`.

This final tuned model significantly improved minority class recall and achieved an overall test accuracy of 87%, reflecting stronger generalization across both classes as shown in Table 5.12.

In addition to overall tuning, we perform targeted hyperparameter searches to prioritize classification performance on individual classes. When optimizing for class 1 (the minority class), the best configuration achieved an F1 score of 0.7122, demonstrating modest improvement through class reweighting and depth control. In contrast, the classifier performs extremely well on class 0, achieving an F1 score of 0.9293 even without special treatment —

Table 5.12: Classification report for the best random forest model after hyperparameter tuning.

Class	Precision	Recall	F1 Score	Support
0 (bad)	0.93	0.90	0.92	1165
1 (good)	0.67	0.77	0.72	313
accuracy			0.87	1478
macro avg	0.80	0.83	0.82	1478
weighted avg	0.88	0.87	0.87	1478

likely due to its larger sample size and greater separation in feature space.

Finally, we evaluate model discrimination using the AUC-ROC metric. The ROC (Receiver Operating Characteristic) curve plots the true positive rate (TPR) against the false positive rate (FPR) across various classification thresholds, revealing the model's ability to rank examples correctly [19, 20]. The AUC (Area Under the Curve) quantifies this: a score of 1.0 indicates perfect separation, while 0.5 suggests random guessing [20]. Our tuned random forest model achieves an AUC of 0.912 — indicating excellent classification performance across thresholds.

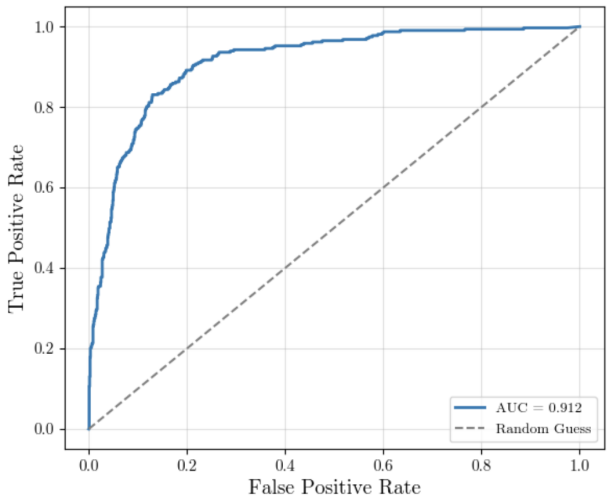


Figure 5.14: ROC-AUC curve for the tuned Random Forest model (with optimal set of hyperparameters), showing a strong area under the curve (AUC = 0.912). This indicates excellent classification performance and consistent ranking ability across decision thresholds.

This high AUC reflects not just strong point predictions, but consistent ranking perfor-

mance across decision thresholds — a key metric in imbalanced classification problems like ours.

To summarize, this final tuned model significantly improved minority class recall and achieved an overall test accuracy of 87%, reflecting stronger generalization across both classes. This result is corroborated by the ROC-AUC score of 0.912, which signals excellent discriminative capability [19]. Note that AUC is a threshold-independent metric that measures the probability the classifier ranks a randomly chosen positive example higher than a negative one [19]. An AUC above 0.90 indicates that the classifier can effectively distinguish between classes even when the decision boundary is varied, making this result especially meaningful in imbalanced settings where accuracy alone can be misleading [19].

That said, performance remains asymmetric. The majority class (bad) exhibits high precision (0.93), high recall (0.90), and a strong F1 score (0.92), reflecting both abundant support (1165 examples) and high separability in feature space. In contrast, the minority class (good) shows improvement but still lags, with a precision of 0.67 and recall of 0.77. Several factors may account for this discrepancy. First, the training set is imbalanced due to the use of a 20th-percentile cost threshold to define "good" outcomes — resulting in class 1 comprising just 20% of the total data [19]. Second, it is possible that the feature distributions for "good" configurations overlap significantly with "bad" ones in certain regions, reducing the model's confidence and increasing false positives. Third, although class weighting and hyperparameter tuning improved recall, further techniques such as synthetic resampling or cost-sensitive learning may be necessary to isolate borderline class 1 regions more effectively [19].

Having established a strong baseline using bagging, we now turn to a second ensemble method: boosting.

5.3.2 Boosting

We follow the same procedure as before, but substitute in a boosting-based model — specifically, XGBoost — to directly compare performance across ensemble strategies. Recall that like random forests, XGBoost is an ensemble of decision trees, but it differs in how those trees are constructed and combined.

To begin, we train an XGBoost classifier and extract gain-based feature importances. Here, *Endcap Voltage* emerges as the most informative feature, followed by *Modulation Amplitude* and *Voltage*. These results are summarized in Table 5.13.

Table 5.13: Feature importance scores from the trained XGBoost classifier.

Feature	Importance (%)
Endcap Voltage	44.57
Modulation Amplitude	31.96
Voltage	23.47

Compared to the feature importances from random forests, this ranking shows a subtle shift in emphasis. This divergence stems from structural differences in how each model learns from data.

Random forests train many independent trees on different subsets of the data and average their outputs. They tend to assign high importance to features that are useful globally — i.e., those that consistently perform well as early splits [19, 20, 22]. In contrast, XGBoost constructs trees sequentially, where each new tree corrects the mistakes of its predecessors. This process prioritizes features that reduce residual errors at specific stages, even if those features aren't universally predictive [24]. As a result, XGBoost tends to emphasize locally informative features and is often more sensitive to subtle feature interactions [19, 24].

We now assess the model's classification performance on the test set. XGBoost achieves an overall accuracy of 87.28%, with particularly strong performance on the majority class. It achieves 0.90 precision and 0.95 recall for the `bad` class, and a lower recall of 0.60 for the `good` class. These results yield a macro F1 score of 0.79 — comparable to the tuned random

forest and reflective of strong generalization despite the class imbalance.

Table 5.14: Classification report for the XGBoost model on the test set.

Class	Precision	Recall	F1 Score	Support
bad	0.90	0.95	0.92	1165
good	0.75	0.60	0.67	313
accuracy			0.87	1478
macro avg	0.82	0.77	0.79	1478
weighted avg	0.87	0.87	0.87	1478

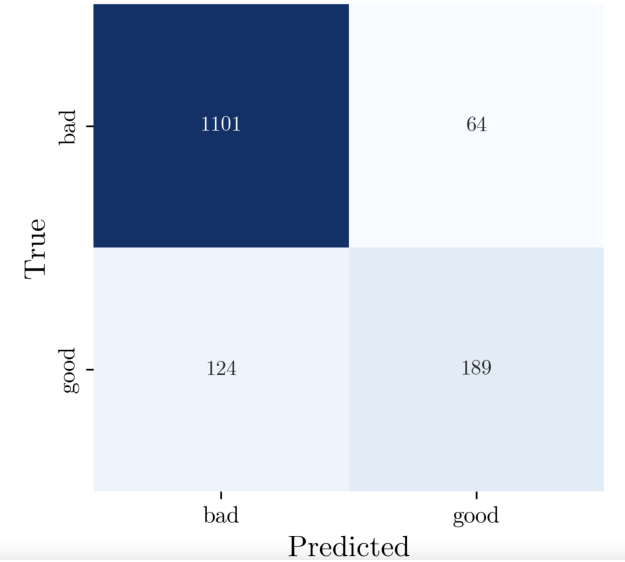


Figure 5.15: Confusion matrix showing performance on the testing set for the initial boosting (XGBoost) model.

To refine performance further, we next explore hyperparameter tuning. As with the bagging model, we apply randomized hyperparameter search to three objectives: class 0 F1, class 1 F1, and macro-averaged F1. Each search samples 300 configurations using 5-fold stratified cross-validation. For class 1 tuning, the best model achieved an F1 score of 0.6862 using 649 estimators, a max depth of 4, a learning rate of 0.046, and a positive class weighting of 2. The class 0–optimized model achieved a notably higher F1 score of 0.9262 with deeper trees and no class weighting. Macro F1 optimization produced a balanced model scoring 0.8049 with parameters closely aligned with the class 1 case. These results reinforce earlier

findings: class 0 is easier to classify, while improving class 1 recall requires careful tuning.

Finally, to benchmark XGBoost against related algorithms, we compare three popular boosting methods: XGBoost, AdaBoost, and Gradient Boosting. All were tuned to maximize macro F1. XGBoost and Gradient Boosting perform comparably well, achieving scores of 0.8002 and 0.8033 respectively. However, AdaBoost falls short with a macro F1 of 0.4645. This underperformance stems from AdaBoost's reliance on shallow learners[17] and its lack of regularization. Without deeper trees or robust error correction, AdaBoost struggles to capture the nonlinear structure of the dissociation problem. This highlights the importance of using expressive models with well-tuned capacity when modeling structured experimental systems.

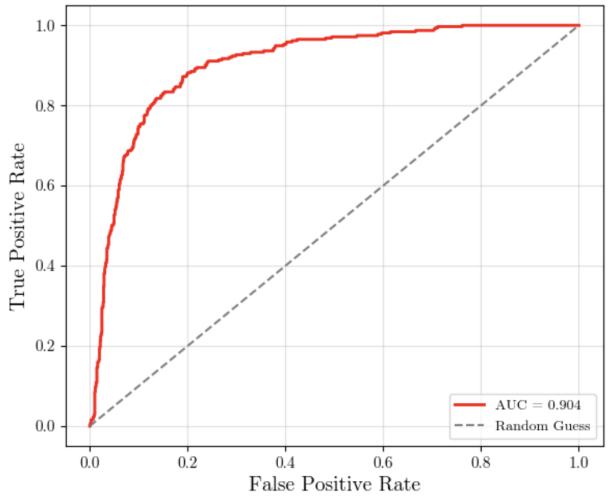


Figure 5.16: ROC-AUC curve for the tuned XGBoost model (with optimal set of hyperparameters), showing a strong area under the curve (AUC = 0.904). This indicates excellent classification performance and consistent ranking ability across decision thresholds, despite being marginally lower than the bagging curve's AUC.

Taken together, these results demonstrate the strengths and limitations of boosting-based methods, with XGBoost and Gradient Boosting offering strong predictive performance and AdaBoost highlighting the risks of underfitting in complex systems. Overall, both Random Forest and XGBoost achieved comparable predictive performance across metrics

[17] A shallow learner is a weak model with limited capacity, such as a decision stump — a one-level tree that makes decisions based on a single feature.

such as accuracy, F1 score, and AUC, suggesting that either model is well suited for this task. The most notable divergence between them lies in their feature importance profiles: Random Forest consistently emphasized RF voltage, while XGBoost gave greater relative weight to endcap voltage and modulation amplitude. This distinction implies that the models capture different structural aspects of the parameter space, and thus offer interchangeable yet complementary perspectives on the classification problem.

Having established the classification performance of these ensemble models, we now shift focus to a complementary approach — one that provides local, interpretable insight into parameter space structure and classification resilience.

5.3.3 Interpolation and KNN Manifold

It is important to emphasize that the interpolation manifold system is still an experimental tool for evaluating local classification stability. As such, the current results should be viewed as preliminary and open to further refinement. Nonetheless, to gain a baseline understanding of its behavior, we compare its performance against a standard K-Nearest Neighbors (KNN) model using 15 neighbors. Specifically, we visualize the distribution of good-to-bad scores produced by each method using side-by-side boxplots. This comparison helps highlight the respective tendencies of each algorithm when estimating local robustness in the parameter space.

Figure 5.17 reveals key differences between the 15-NN and interpolated neighbor scoring methods. While both exhibit low medians near zero, the 15-NN approach has a much wider spread, with several scores extending to high values. This suggests that 15-NN can occasionally overestimate local robustness due to isolated "good" points in sparse regions. In contrast, the interpolated scores are more tightly clustered, with a lower upper bound and reduced variance. This reflects the method's smoother, geometry-aware evaluation of local structure, which avoids overreacting to noisy neighbors. Overall, the interpolated score appears to offer a more stable and cautious estimate of neighborhood quality, especially in

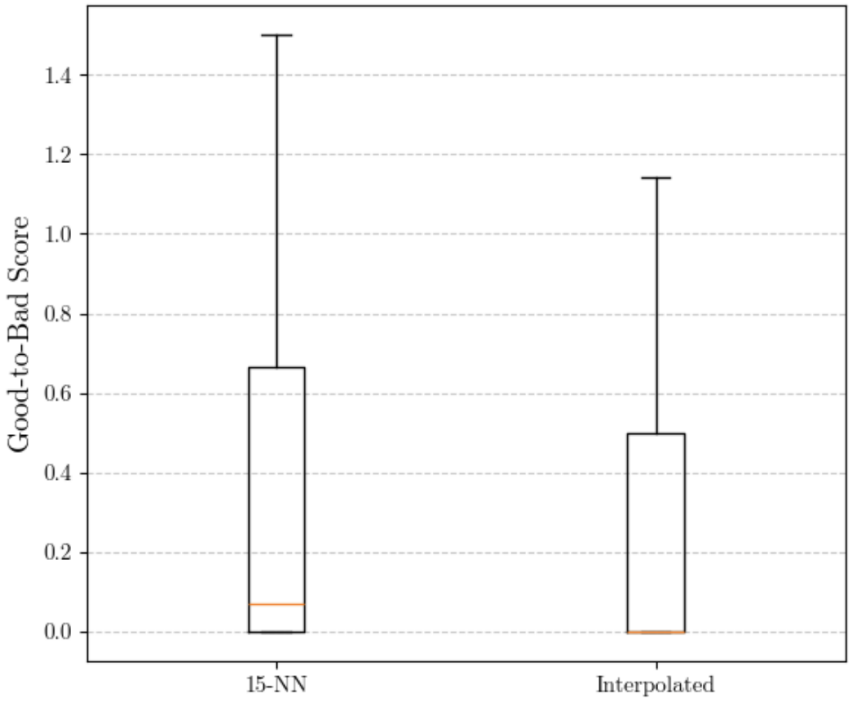


Figure 5.17: Boxplot comparing the distributions of good-to-bad scores from the interpolated manifold model and a traditional 15-nearest neighbors model across 500 randomly selected points. Outliers are hidden for clarity.

uncertain or boundary regions.

The interpolation-based neighbor scoring framework provides several principled advantages over standard K-nearest neighbors (KNN) when estimating the local robustness of a configuration. First, unlike KNN, which treats all neighbors equally, interpolation assigns influence based on spatial continuity, allowing it to capture fine-grained changes in local geometry and system behavior. Second, by constructing concentric hyperspherical shells and sampling synthetic points, the method probes not just along observed datapoints but also within unmeasured regions of the parameter space — offering a richer, more continuous understanding of local dynamics. Third, RBF interpolation introduces smoothness by design, reducing the sensitivity to sampling density and outliers that KNN is prone to. Fourth, the framework blends real observations and interpolated estimates, preserving empirical grounding while filling in gaps in the data manifold — a key benefit in high-dimensional or sparse

datasets. Finally, this approach quantifies not just how similar a configuration is to known "good" ones, but how stable its classification remains under small, local perturbations — a form of robustness KNN alone cannot assess. Together, these properties make interpolation-based scoring a more flexible, informative, and physically interpretable tool for configuration evaluation in noisy, complex experimental systems.

Taken as a whole, these findings underscore the value of combining global classification with localized interpolation-based scoring to form a more nuanced picture of configuration quality. The methods introduced here are not only effective in identifying high-performing regions of parameter space, but also in flagging fragile or borderline configurations that warrant further scrutiny. With this foundation in place, we now turn to potential extensions — outlining next steps for scaling, refining, and applying the broad framework we have laid out to future experiment and exploration.

Chapter 6

Future Work and Next Steps

While the machine learning framework developed in this thesis already yields promising insights and optimization strategies, there remain numerous opportunities to expand its scope and deepen its impact. This chapter outlines several concrete directions for future work, each of which builds on the foundations laid here while pointing toward a more comprehensive, physically grounded, and experimentally integrated modeling framework.

6.1 Expansion of Feature Set in Training Data

The most immediate next step is to broaden the set of input features used in model training. Although practical constraints such as simulation time and data dimensionality will always be relevant, all methods deployed here — ensemble classifiers, Monte Carlo sampling, and interpolative scoring — are designed to scale effectively in high-dimensional feature spaces [19]. By incorporating additional features, such as detector tuning, experimental sequence, or modulation waveform metadata, the model may capture subtle patterns that bolster the current model. The long-term goal is to boost predictive performance across stricter optimization regimes — ideally improving precision and recall for "good" classifications, even under stringent[1] evaluation criteria.

[1]By *stringent* here, we mean cost functions that are extremely selective, such as a 5th percentile cutoff.

6.2 Cost Function Design

Another rich area for exploration lies in the design of alternative cost functions. The slope–R^2 composite metric introduced in this work provides a solid baseline, but it represents only one lens through which dissociation quality can be measured. Future efforts could use symbolic regression or genetic programming to search for mathematical structures that more directly reflect high-resolution dissociation behavior [20]. One promising avenue involves defining trace-based heuristics — such as total drop magnitude, count of monotonic segments, or temporal entropy — and constructing nonlinear functions of these quantities. Alternatively, adaptive thresholds and stage-specific penalties may allow the model to flexibly tune its objective across different dissociation regimes. Several such cost function variants are outlined in Appendix A.

6.3 Interpolation Manifold

The interpolation-based scoring system introduced in this thesis offers a compelling new way to assess local robustness, but there is substantial room to develop this further. One direction involves refining how synthetic neighbors are sampled — for example, by biasing toward high-density regions or adapting sphere radii to local curvature. Another possibility is to incorporate kernel-based smoothing or uncertainty-aware interpolation methods [24], which would allow the scoring function to explicitly reflect confidence in under-sampled regions. Dimensionality reduction techniques such as t-SNE or UMAP may also prove useful here, offering a means to visualize the global structure of the parameter space and highlight transitions between stable and unstable zones [27, 28]. Ultimately, the goal is to improve both interpretability and reliability in edge-case classifications. Beyond structural analysis, it is also important to examine the assumptions that underpin the model's treatment of noise and uncertainty.

6.4 Empirical Verification of Uniform Stochasticity

One assumption underlying our modeling framework is that stochastic variation in photon scattering is stationary across various parameter sets and is reasonably approximated by repeated sampling of the same configuration. However, this assumption remains largely empirical. Future experimental campaigns should focus on collecting detailed dissociation traces — ideally across a wide range of experimental conditions — in order to construct formal distributions for count noise. These distributions could be modeled parametrically (e.g., as Poisson or Gaussian mixtures) or non-parametrically (e.g., via kernel density estimation), and used to better inform simulation fidelity and Monte Carlo variability. Validating these stochastic assumptions is essential for the trustworthiness of the model's uncertainty estimates.

6.5 Closing Thoughts

Finally, and perhaps most critically, this framework should be applied to live experiments in the lab. The classification and optimization strategies developed here were specifically designed to interface seamlessly with real-world experimentation, and deployment on physical systems is the clearest test of their value. It's important to stress, however, that any mismatch between simulated and experimental results is not a failure — it's information. This model was built to learn. Disagreement provides the examples needed for refinement, and as the learner absorbs more data from the physical system, its predictions will only improve. This is the strength of a data-driven machine learning framework: it is not rigid, but adaptive and self-correcting.

Moreover, in practice, integrating all of this into an interactive dashboard should be straightforward given the current codebase. Automated job scheduling with `sbatch` array scripts would allow new simulation points to be continuously generated and folded into the dataset, enabling the model to update and improve on a daily basis.

This thesis has introduced a machine learning-guided framework for analyzing and optimizing number resolution in trapped ion systems, supported by simulation-derived data and statistically grounded scoring metrics. While the models developed here are designed around the specific challenges of dissociation detection, the broader methodology — combining simulation, probabilistic reasoning, and adaptive classification — is widely applicable to other experimental contexts where noisy signals and complex parameter spaces pose obstacles to interpretability and control. Our hope is that this work offers not only immediate utility for tuning dissociation behavior in precision ion experiments, particularly in Be^+–O^+ trapping platforms like those used in our lab, but also a generalizable blueprint for applying machine learning to experimental systems in ion trapping and physics simulations more broadly. Of course, many aspects of the framework remain open for refinement. But the foundation laid here — in both code and concept — marks a meaningful step toward more autonomous, data-driven experimentation in precision ion trapping platforms. In this sense, the thesis is not only a conclusion — it is an invitation to iterate, expand, and deploy.

Appendix A

Alternative Cost Functions

This appendix presents a set of alternative cost functions designed to evaluate dissociation trace quality in different ways. These cost functions incorporate not only the slope and R^2 values from the trace, but also probabilistic classification confidence p, neighborhood stability scores s, and structural features of the scattering signal such as monotonicity and variance.

Simple and Moderately Nonlinear Cost Functions

Cost Function 1: Weighted Additive (Baseline Style)

$$\text{Cost}_1 = 0.7 \cdot \texttt{Slope\_norm} + 0.2 \cdot \texttt{R-sq\_norm} + 0.1 \cdot (1 - p) \cdot 100 \tag{A.1}$$

Justification: This function extends the original baseline by including classification confidence. It keeps the form additive and interpretable, with soft penalties for low-confidence predictions.

Cost Function 2: Nonlinear Stability Wrapping

$$\text{Cost}_2 = \frac{0.8 \cdot \texttt{Slope\_norm} + 0.2 \cdot \texttt{R-sq\_norm}}{1 + 2s^2} \tag{A.2}$$

Justification: Strongly penalizes unstable regions of parameter space via a squared stability score. The cost drops sharply in well-behaved, high-stability regions.

Cost Function 3: Exponential Penalty for Weak Drops

$$\text{Cost}_3 = \left(\frac{1}{n} \sum_{i=1}^{n-1} e^{-\Delta_i/100} \right) \cdot 100 \tag{A.3}$$

where $\Delta_i = \texttt{Scattering}_i - \texttt{Scattering}_{i+1}$

Justification: Penalizes small changes in scattering. Larger drops contribute exponentially less to the cost, rewarding clear dissociation transitions.

Cost Function 4: Hybrid Rank-Based Penalty

$$\text{Cost}_4 = \left(\frac{\texttt{Slope\_norm}^{1.3} + \texttt{R-sq\_norm}^{1.3}}{2} \cdot (1-s)^2 \cdot (1-p) \right)^{0.5} \tag{A.4}$$

Justification: Strongly punishes configurations with poor slope or R^2, particularly when classification and neighborhood stability are also low. Useful in noisy datasets.

Cost Function 5: Monotonicity-Aware Model

Let $M \in [0, 1]$ be the fraction of strictly decreasing transitions in the scattering trace.

$$\text{Cost}_5 = (0.6 \cdot \texttt{Slope\_norm} + 0.4 \cdot \texttt{R-sq\_norm}) \cdot \left(1 + e^{-10(M-0.6)}\right) \tag{A.5}$$

Justification: Introduces a sigmoid-style penalty that harshly punishes traces with low monotonicity. Rewards sharply decreasing traces with minimal reversals.

Complex Cost Functions

Cost Function 6: Composite Interaction Map with Local Variance Suppression

Let V_{local} be the variance of the 5 central scattering values, and let $\Delta_{\max}$ be the largest absolute jump across all steps.

$$\text{Cost}_6 = \left(\frac{(\texttt{Slope\_norm} + \texttt{R-sq\_norm})^2}{1 + 3s^2 \cdot p} \right)^{0.75} + \log(1 + V_{\text{local}}) + \frac{50}{1 + \Delta_{\max}} \tag{A.6}$$

Justification: Fuses multiple effects: local variance suppression, maximum drop size, and nonlinear penalty reduction through classification confidence and neighborhood stability.

Cost Function 7: Piecewise Entropy-Weighted Composite with Penalized Variability Envelope

Let $\mathbf{y} = \{y_1, \ldots, y_{10}\}$ be the scattering values, and define:

$$\text{Entropy} = -\sum_{i=1}^{9} \hat{p}_i \log \hat{p}_i, \quad \text{where } \hat{p}_i = \frac{|y_{i+1} - y_i|}{\sum_{j=1}^{9} |y_{j+1} - y_j|}$$

$$\text{Cost}_7 = \begin{cases} (\texttt{Slope\_norm}^{0.9} + 3 \cdot (1 - s)^{1.5}) \cdot \log(1 + \text{Entropy}) & \text{if } M < 0.8 \\ (\texttt{R-sq\_norm}^{1.1} + 10 \cdot (1 - p)^2) \cdot \left(1 + \frac{\sigma_y}{\Delta_{\text{avg}}^2}\right) & \text{otherwise} \end{cases} \tag{A.7}$$

Justification: Captures signal entropy and penalizes jump irregularity. It changes form depending on monotonicity threshold, making it highly adaptable to structural variation.

Cost Function 8: Exponential-Logarithmic Hybrid with Drop Rank Scoring and Saturation Envelope

Let R_{rank} be the trace's rank among all samples based on average drop magnitude.

$$\text{Cost}_8 = 100 \cdot \left[1 - \tanh \left(\frac{s \cdot p \cdot \log(1 + \Delta_{\text{avg}})}{1 + \left(\frac{\texttt{Slope\_norm} \cdot \texttt{R-sq\_norm}}{R_{\text{rank}}} \right)^{0.7}} \right) \right]^{1.2} \tag{A.8}$$

Justification: Highly nonlinear formulation that sharply differentiates between elite and average traces. Strong penalty for high-ranked but noisy or unstable configurations.

Appendix B

SLURM Usage for qlicS_batch

This appendix provides a concise walkthrough of common SLURM commands useful for running and debugging `qlicS_batch` jobs on a high-performance computing cluster. It starts from interactive testing via `srun` and culminates in full-scale job submission via `sbatch`.

Interactive Sessions:

- `srun --pty bash`

  Launch a basic interactive shell on a compute node. Useful for debugging and loading modules.

- `srun -n 4 --pty bash`

  Launches an interactive shell with 4 tasks (useful when testing multiprocessing or MPI).

- `srun -n 50 --mem=500G --time=00:10:00 --pty bash`

  Starts an interactive job with 50 processes, 500 GB of memory, and a wall time limit of 10 minutes. Ideal for short batch-wide testing of ‘qlicS_batch‘.

- `srun -n 1 python my_script.py`

  Executes a Python script using 1 process.

- `srun -n 32 python qlicS_batch.py --config-folder config_dir --threads 32`

  Example call to execute ‘qlicS_batch.py‘ with 32 parallel threads using ‘srun‘.

Submitting Jobs via `sbatch`:

Use `sbatch` when running larger or longer jobs. Below is a minimal working example of a SLURM batch script for running `qlicS_batch`:

Listing B.1: Example SLURM job script: submit_qlics.sh

```
#!/bin/bash
#SBATCH --job-name=qlics_batch
#SBATCH --output=logs/output_%j.txt
#SBATCH --error=logs/error_%j.txt
#SBATCH --ntasks=64
#SBATCH --mem=750G
#SBATCH --time=12:00:00
#SBATCH --partition=general
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=your_email@domain.edu
```

Submit the job using:

```
sbatch submit_qlics.sh
```

Useful Commands for Monitoring Jobs:

- `squeue -u $USER` — show your current jobs
- `scancel <job_id>` — cancel a submitted job
- `sacct -j <job_id>` — view job accounting details

This workflow provides both an interactive and automated pipeline for efficiently launching `qlicS_batch` at scale.

Appendix C

Annotated Configuration File

Below is a fully annotated example configuration file, formatted for clarity using the `minted` package. This file corresponds to a typical input used by QLICS to define simulation parameters, trap geometry, laser parameters, and the experimental sequence. Comments are included inline to clarify the purpose of each parameter in the context of the simulation [3, 15]:

```
[directory]
dump_dir = /Users/michaelmitchell/qlicS/data/2024-07-01_17-08-03/  # Output path
↪  for simulation data

[live_vars]
current_timesequence_pos = 2  # Time index used in iter mode; selects timestep
↪  chunk from list

[constants]  # Physical constants used in LAMMPS unit conversions
h = 6.626e-34
c = 299792458
amu = 1.6605402e-27
ele_charge = 1.60217663e-19
boltzmann = 1.380649e-23
```

```
[ions]  # All ion species in the simulation with physical properties
be+ = [{'mass': 9, 'charge': 1}, {'natural linewidth': 113097335.53, 'absorption
↪  center': 9.578e14, 'saturation': 765}]
o2+ = [{'mass': 32, 'charge': 1}, {'natural linewidth': None, 'absorption
↪  center': None, 'saturation': None}]
o+ = [{'mass': 16, 'charge': 1}, {'natural linewidth': None, 'absorption center':
↪  None, 'saturation': None}]

[sim_parameters]
log_steps = 10  # Number of simulation steps per log output
timesequence = [[1e-08, 2.5e5],[1e-08, 2.5e5],[1e-08, 2.5e5],[1e-08, 2.5e5]]  #
↪  Each timestep and duration pair
lammps_boundary_style = ['f', 'f', 'f']  # Fixed boundary conditions
lammps_boundary_locations = [[-.001, .001],[-.001, .001],[-.001, .001]]  # Trap
↪  size bounds (m)
lammps_allow_lost = True  # Allow ions to leave the simulation box

[detection]
detection_timestep_seq = [[]]  # Timestep windows during which photon counts are
↪  collected
detector_area = 0.0001  # Area of detector (m^2)
detector_effeciency = 0.01  # Detection efficiency
detector_distance = 0.2  # Distance from center to detector (m)

# Three ion clouds, each with a species, radius (m), and number of ions
[ion_cloud_0]
uid = 1
species = be+
```

```
radius = 1e-4
count = 20

[ion_cloud_1]
uid = 2
species = o2+
radius = 1e-4
count = 9

[ion_cloud_2]
uid = 3
species = o+
radius = 1e-4
count = 1

# RF trap definitions (1 per cloud), with voltages and axial confinement settings
[trap_0]
uid = 11012277363
target_ion_pos = 0
radius = 1.25e-3
length = 1.5e-3
kappa = 0.17
frequency = 11040000
voltage = 50
endcapvoltage = 1
pseudo = True  # Use pseudopotential approximation

[trap_1]
uid = 12012277363
```

```
target_ion_pos = 1
radius = 1.25e-3
length = 1.5e-3
kappa = 0.17
frequency = 11040000
voltage = 50
endcapvoltage = 1
pseudo = True

[trap_2]
uid = 13012277363
target_ion_pos = 2
radius = 1.25e-3
length = 1.5e-3
kappa = 0.17
frequency = 11040000
voltage = 50
endcapvoltage = 1
pseudo = True

# Doppler cooling laser applied to Be+ ions
[cooling_laser_0]
uid = 569202603907002
target_ion_pos = 0
target_ion_type = be+
beam_radius = 0.0001
saturation_paramater = 100
detunning = -300000000.0  # Red-detuned for cooling
laser_direction = [0.5, 0.5, 0.7071]
```

```
laser_origin_position = [0, 0, 0]

# Resonant laser used to induce scattering on Be+ (visible readout)
[scattering_laser]
scattered_ion_indices = [0, 20]
target_species = be+
laser_direction = [-0.5, -0.5, -0.7071]
saturation_paramater = 5
frequency = 957800000000000.0

# Modulation/tickle field applied to drive resonance motion
[modulation_0]
uid = 469202603907006
amp = 1  # Field amplitude
frequency = 6.5e5  # Sweep center frequency
ex0 = 1  # Field aligned in x-direction
# All other spatial field components off
x_shift = 0
y_shift = 0
z_shift = 0
static = [0, 0, 0]

[iter]  # Defines param scan for iter-based simulations
scan_objects = ["cooling_laser_0","modulation_0"]
scan_var = ["modulation_0", "frequency"]
scan_var_seq = [999000, 178000, 387000, 999000]
iter_timesequence = [[1e-08, 1e5], [1e-08, 1e5], [1e-08, 1e5]]
iter_detection_seq = [[2.9e5, 3.0e5]]
```

```
com_list =
↪  cooling_laser,evolve,r_569202603907002,tickle,evolve,r_469202603907006,evolve
↪  # Defines command sequence

[exp_seq]  # Startup sequence before each iteration
com_list = dumping,cloud,cloud,cloud,trap,trap,trap,iter
```

This configuration file governs all aspects of a simulation run in QLICS, including crystal generation, trap field behavior, tickling experiments, and photon scattering detection. Each section is modular and easily editable for experimental prototyping or automated optimization.

Bibliography

[1] A. Lunstad, *Driving Forbidden Vibrational Transitions in Molecular Oxygen*, Undergraduate thesis, Amherst College (2021).

[2] A. Hartman, *Two-Photon Vibrational Transitions in* O_2^+, Undergraduate thesis, Amherst College (2022).

[3] M. Mitchell, *A Software Package for the Simulation and Optimization of Trapped Ion Experiments*, Undergraduate thesis, Amherst College (2024).

[4] A. P. Singh, M. Mitchell, W. Henshon, A. Hartman, A. Lunstad, B. Kuzhan, and D. Hanneke, "State Selective Preparation and Nondestructive Detection of Trapped (O_2^+)," `http://arxiv.org/abs/2410.14832` (2024), submitted for publication.

[5] W. Henshon, *Radiofrequency Circuit Design for Ion Trapping of* O_2^+ *Molecules*, Undergraduate thesis, Amherst College (2023).

[6] S. Qiao, *Constructing a Linear Paul Trap System for Measuring Time-variation of the Electron-Proton Mass Ratio*, Undergraduate thesis, Amherst College (2013).

[7] C. Pluchar, *An Ultraviolet Laser for Beryllium Photoionization*, Undergraduate thesis, Amherst College (2018).

[8] R. A. Carollo, D. A. Lane, E. K. Kleiner, P. A. Kyaw, C. C. Teng, C. Y. Ou, S. Qiao, and D. Hanneke, "Third-harmonic-generation of a diode laser for quantum control of beryllium ions," Optics Express **25**, 7220 (2017).

[9] H. J. Metcalf and P. van der Straten, *Laser Cooling and Trapping* (Springer, 1999).

[10] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," Applied Physics Reviews **6**, 021314 (2019).

[11] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, *et al.*, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," Computer Physics Communications **271**, 108171 (2022).

[12] D. Hanneke, B. Kuzhan, and A. Lunstad, "Optical clocks based on molecular vibrations as probes of variation of the proton-to-electron mass ratio," Quantum Science and Technology **6**, 014005 (2020).

[13] B. Nourse and R. Cooks, "Aspects of recent developments in ion-trap mass spectrometry," Analytica Chimica Acta **228**, 1 (1990).

[14] P. H. Dawson, *Quadrupole Mass Spectrometry and Its Applications* (Elsevier, 2013).

[15] M. Mitchell, "qlicS," `https://github.com/mmitch12202/qlicS` (2024).

[16] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson, "A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters," The Journal of Chemical Physics **76**, 637 (1982), ISSN 0021-9606.

[17] M. F. Andersen, C. Ryu, P. Cladé, V. Natarajan, A. Vaziri, K. Helmerson, and W. D. Phillips, "Quantized Rotation of Atoms from Photons with Orbital Angular Momentum," Phys. Rev. Lett. **97**, 170406 (2006).

[18] A. Wagaman and L. Spector, "Acquisition of High Performance Computing System for

Interdisciplinary Research and Teaching," National Science Foundation Grant #2117377 (2021).

[19] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning* (Springer, 2017), 2nd ed.

[20] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning* (MIT Press, 2018), 2nd ed.

[21] M. Rupp, "Machine Learning for Quantum Mechanics in a Nutshell," International Journal of Quantum Chemistry (2015).

[22] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms* (Chapman and Hall/CRC, 2012).

[23] R. Polikar, *Ensemble Learning* (Wiley-IEEE Press, 2021).

[24] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (O'Reilly Media, 2019), 2nd ed.

[25] P. I. Frazier, "A Tutorial on Bayesian Optimization," arXiv preprint (2018), `https://doi.org/10.48550/arXiv.1807.02811`.

[26] P. Mertikopoulos, H. Zenati, V. Chandrasekhar, and G. Piliouras, "On the Almost Sure Convergence of Stochastic Gradient Descent in Non-Convex Problems," arXiv preprint arXiv:2006.11144 (2020).

[27] M. D. Buhmann, *Radial Basis Functions: Theory and Implementations* (Cambridge University Press, 2003).

[28] R. L. Hardy, "An Algorithm for Selecting a Good Value for the Parameter c in Radial Basis Function Interpolation," Computational Statistics **16**, 303 (2001).

[29] R. W. Shonkwiler and F. Mendivil, *Explorations in Monte Carlo Methods* (Springer, New York, 2013), ISBN 9781461470184, URL https://link.springer.com/book/10.1007/978-1-4614-7018-4.

[30] M. R. Hush, "M-LOOP Documentation," https://m-loop.readthedocs.io/en/stable/ (2024), accessed: 2024-12-17.

[31] P. Wigley, P. Everitt, A. van den Hengel, *et al.*, "Fast machine-learning online optimization of ultra-cold-atom experiments," Scientific Reports **6**, 25890 (2016).

[32] Z. Zhou, Y. S. Ong, P. B. Nair, A. J. Keane, and K. Y. Lum, "Combining Global and Local Surrogate Models to Accelerate Evolutionary Optimization," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **37**, 66 (2007).

[33] D. C. Wilson and B. A. Mair, "Thin-Plate Spline Interpolation," in "Sampling, Wavelets, and Tomography," , edited by J. J. Benedetto and A. I. Zayed (Birkhäuser, Boston, MA, 2004), Applied and Numerical Harmonic Analysis, ISBN 978-1-4612-6495-8, URL https://doi.org/10.1007/978-0-8176-8212-5_12.

[34] G. Bonaccorso, *Mastering Machine Learning Algorithms* (Packt Publishing, Birmingham, UK, 2018), ISBN 9781788621113.

[35] W. Paul, "Electromagnetic traps for charged and neutral particles," Rev. Mod. Phys. **62**, 531 (1990), URL https://doi.org/10.1103/RevModPhys.62.531.

[36] C. Chen, D. H. Zanette, D. Czaplewski, S. Shaw, and D. López, "Direct observation of coherent energy transfer in nonlinear micro-mechanical oscillators," arXiv preprint arXiv:1612.00490 (2016), URL https://arxiv.org/abs/1612.00490.

[37] D. A. Baiko, D. G. Yakovlev, H. E. D. Witt, and W. L. Slattery, "Coulomb crystals in the harmonic lattice approximation," Physical Review E **61**, 1912 (2000), URL https://arxiv.org/abs/physics/9912048.

[38] A. Kumar, R. Singh, and P. Sharma, "Solutions of the Mathieu–Hill Equation for a Trapped-Ion Harmonic Oscillator: A Qualitative Discussion," Mathematics **12**, 2963 (2024), URL https://www.mdpi.com/2227-7390/12/19/2963.