Erasmus+

PSRS

Erasmus Mundus Joint Master degree
Photonics for Security Reliability and Safety

Master Photonics for Security Reliability and Safety (PSRS)

Université Jean Monnet Saint-Étienne

University of Eastern Finland

UPEC Université Paris-Est Créteil Val de Marne

Politecnico di Torino

Neural Network Segmentation of Charge Stability Diagrams for the Auto-Tuning of Silicon Quantum Dots for Spin Qubits

Master Thesis Report

Presented by

Peter Samaha

and defended at

University Jean Monnet

3 September 2025

Academic Supervisor(s):
Prof. Carlo Ricciardi
Host Supervisors:
Yann Beilliard,
Pierre-André Mortemousque

Jury Committee:
Prof. Carlo Ricciardi, Polytechnic University of Turin
Prof. Amine Naït Ali, Paris-East Créteil University

Abstract

Automatic tuning of gate-defined semiconductor Quantum Dots (QDs) is a key bottleneck on the path toward scalable qubit architectures. In this thesis, we develop and validate a Machine Learning (ML)-driven pipeline for offline and prospective online charge state auto-tuning, using Charge Stability Diagrams (CSDs) to locate the single charge regime. We assemble and manually annotate a large dataset of CSD images from nine distinct device designs fabricated across multiple process batches and patterned on different wafers and die locations. A U-Net–based Convolutional Neural Network (CNN) is trained to segment charge transition lines under challenging, low-contrast cryogenic conditions and measurement noise. Through five-fold cross-validation, our model achieves a **success rate of 80.0% in locating the single charge regime** tested on a total of **1015 CSDs**. The highest-performing device designs were **Design D and E** with success rate of **88%** tested on 147 and 138 stability diagrams respectively. Considering mask-level performance, **Mask I** achieved 589/695 (84.7%) while **Mask II** achieved 223/320 (69.7%). Detailed failure analysis highlights common modes such as missed, faint, spurious, and fragmented lines, and motivates solutions for these cases. We outline a roadmap for real-time integration in a cryogenic wafer prober, on-chip cryostat deployment, and multi-qubit scaling via joint segmentation and physics-guided postprocessing. Our results demonstrate that data-driven semantic segmentation can reliably automate charge tuning, paving the way for closed-loop control protocols essential to fault-tolerant quantum computing.

Contents

Contents ii

List of Figures v

List of Tables xi

1 Introduction 1

2 State of the Art 11

2.1 Fundamentals of Quantum Computing 11

2.1.1 The Challenges of Quantum Computers 11

2.1.2 Physical Platforms to Realize a Qubit 12

2.2 Quantum Dots 12

2.2.1 Quantum Confinement 12

2.2.2 Quantum Dot Realizations 13

2.2.3 Electrostatically Defined Semiconductor Quantum Dots 14

2.3 Quantum Dots for Qubit Realization 17

2.4 Coulomb Blockade in QD 18

2.5 Readout Method: Single Electron Transistor 21

2.6 Charge Stability Diagram 22

2.7 Machine Learning 22

2.7.1 Learning Paradigms 24

2.7.2 Data Types 25

2.7.3 What Is a Machine Learning Model 26

2.7.4 Training Process of a Neural Network 28

2.7.5 Transfer Learning and Fine-Tuning 29

2.7.6 Evaluation 30

2.7.7 Model Inference 33

2.8 Conclusion 33

3 The Tuning Process of Quantum Dots **34**
3.1 Tuning of QDs . . . 34
3.2 Charge Tuning of QDs . . . 35
3.3 Literature Review and State-of-the-Art . . . 38
3.3.1 Classical Heuristic Methods . . . 38
3.3.2 Machine Learning Methods . . . 39
3.4 Proposed Full-Diagram Charge Tuning Method . . . 41
3.5 Device Variability and ML-Enabled Feedback . . . 43
3.6 Conclusion . . . 44

4 Data Acquisition and Preprocessing **46**
4.1 Experimental Setup and Data Acquisition . . . 46
4.2 Data Labeling . . . 49
4.2.1 Type of Transition Lines . . . 50
4.2.2 Labeling Workflow . . . 50
4.3 Data Processing . . . 52
4.3.1 Statistical Feature Extraction . . . 52
4.3.2 Normalization . . . 53
4.3.3 Mask Generation . . . 54
4.3.4 Single Charge Regime Mask Generation . . . 54
4.4 Conclusion . . . 55

5 Implementation of the ML-Based Charge Tuning Pipeline **56**
5.1 Problem Formulation . . . 56
5.2 Model Architecture . . . 57
5.2.1 Encoder: MobileNetV2 Backbone . . . 58
5.2.2 Decoder: Upsampling, Skip Connections, and Feature Fusion . . . 59
5.2.3 Final Prediction Layer . . . 59
5.2.4 Loss Function: Dice Loss . . . 60
5.3 Training Procedure . . . 60
5.3.1 Computational Environment . . . 60
5.3.2 Data Splitting & Cross-Validation . . . 60
5.3.3 Input Preprocessing: Resize and Padding . . . 61
5.3.4 Data Augmentation . . . 63
5.4 Prediction and Postprocessing . . . 63
5.5 Evaluation Metrics . . . 68
5.6 Conclusion . . . 69

6 Experimental Results of Offline Auto Tuning **70**
6.1 Results and Discussion 70
6.1.1 Qualitative Results 70
6.1.2 Quantitative Results 71
6.2 Failure Analysis 75
6.3 Results summary 83

7 Discussion and Future Work **84**
7.1 In-Situ Auto-Tuning in a Cryogenic Wafer Prober 84
7.2 On-Chip Cryogenic Implementation 85
7.3 Scalability to Multi-Qubit Arrays and New Materials 85
7.4 Physics-Guided Feature Extraction 86
7.5 Enhanced Multi-Channel Input Representations 86

8 Conclusion and Perspectives **88**

A Training Details and Metrics **91**

B Offline Auto-Tuning Test Results **97**

Bibliography **100**

List of Figures

1.1 Different physical implementations of a qubit: advantages and disadvantages, and most prominent companies and research lab working on each type [29]. 7

1.2 Simplified 3D depiction of a silicon-on-insulator nanowire field-effect transistor featuring dual-gate structures. (Adapted from [34]) 8

2.1 Schematic representation of quantum confinement and the discretization of energy spectra as the dimension gets reduced. (Taken from [49]) . . . 13

2.2 Schematic of silicon-based structures for the realization of QDs. First column: Schematic of the structure. Second column: Confinement potential of the electrons in the material. Third column: Schematic of the structure representing the source, drain, and gate electrodes. Fourth column: Potential well electrostatically coupled to gate electrodes that can change the electrochemical potential of the electron relative to the source and drain which are tunnel-coupled to the well. (Reproduced from [55]) 15

2.3 **Silicon nanowire CMOS.** (a) Schematic representation of a silicon nanowire (red) on top of an oxide (green) with a gate electrode patterned on the silicon channel (gray). (b) sem top-view image of a silicon nanowire cmos transistor, etched on top of a box in a fd-soi structure. Metallic split-gates are patterned on top of the nanowire which is connected to a source and a drain having the role of electron reservoirs (Taken from [56]). 16

2.4 Tunable single qd potential profile which is formed by applying a bias voltage V_{SD} between source and drain, and a gate voltage V_G on the plunger gate denoted by P. The qd is formed between two barriers denoted B which can be tuned by changing the voltage on the barrier gates which affects the tunneling rate between the source/drain and the dot. 17

2.5 Schematics of the electrochemical potential levels of a qd in the low bias regime (a and b), and the resulting one-dimensional trace of Coulomb peaks and blockades (c). For an applied bias voltage V_{SD}, a small window opens to allow charge flow between source and drain (a). By changing the voltage V_{G} of the plunger gate of the dot, we can manipulate the electrochemical potential of that dot, and by positioning an energy level μ_N of the dot between that of the source and drain (μ_S and μ_D respectively), current will flow from source to drain though the dot and therefore single-electron tunneling will take place between $N-1$ and N (b) which can be visualized as Coulomb peaks in the current vs gate voltages plot (c). When this electrochemical energy of the dot μ_N resides out of the bias window, no current will flow from source to drain and the dot is in the Coulomb blockade regime where the number of electrons is fixed at $N-1$. The tunnel rate between the dot and the reservoirs (Γ_S and Γ_D) dictate the magnitude of the current. (Adapted from [28]) 20

2.6 **Readout Method for the Silicon nanowire CMOS.** (a) Schematic representation of a silicon nanowire with the top gates being used as a readout Single Electron Transistors (SETs) for sensing, and the bottom gates to form few-charge qd to host qubits. (b) Similar device under operation, where we can see the few-charge being formed under the gate with voltage V_{QD} and many-charge set formed under the gate with bias V_{SET}. 21

2.7 **Charge Stability Diagram.** A 2D current map of a qd where the x and y axes represents the voltage sweep applied to the gates, and the color bar represents the measured current value in nA at every point in the map. The numerals represent the number of trapped electrons in the dot, separated by vertical charge transition lines. We can also see the Coulomb oscillations as oblique peaks represented here by the current color bar. The line between the single electron regime and the no charge regime in this case look slightly different from the rest, it is what we refer to as stochastic transition line and it is due to the tunneling rate of the electron between the dot and the reservoir being higher than the sampling rate used for taking the measurement (see Sec. 4.2.1) 23

2.8 Venn diagram showing overlapping relationships between Artificial Intelligence, Machine Learning, Deep Learning, and Computer Vision. 24

2.9 The main steps of supervised learning. 25

2.10 Schematic of feedforward and error backpropagation processes for a single neuron j. The neuron performs the computation in Eq. 2.1 and is connected to an input layer which is an aggregation of input neurons x (in green), and a single output neuron y (in blue) which is the prediction of the model (Adapted from [68]) 26

2.11 Schematic of the architecture of a multilayer ann with error backpropagation, showing an input layer, two hidden layers, and an output layer. Each node represent a neuron (Adapted from [68]) 27

2.12 A schematic representation of a simple cnn architecture showing the convolution layer, pooling layer, and the fully connected layer (Adapted from [69]). 28

2.13 Visual representation of Precision and Recall (Adapted from [77]). 31

2.14 Schematic representation of the Dice metric. 32

2.15 Schematic diagram of the iou metric. 32

4.1 Wafer Cryogenic Prober used to acquire the CSDs data. 47

4.2 Distribution of our CSDs data across mask designs, device polarity, batches, wafers, and device design respectively from top left to bottom right . . . 48

4.3 The labeling process of stability diagrams 51

4.4 Visual representation of the three statistical features (mean, median, and standard deviation) which were calculated from the measurement samples. Each subplot contains a single channel. For better visualization we did not plot them in grayscale. By stacking these features together we get a single three-channel image containing a statistical representation of all measurements. 53

4.5 Ground-Truth mask generated from the manual labeling done on Fig. 4.3 54

5.1 **End-to-end offline-tuning pipeline.** We start with current data measurement (in purple) by sweeping the voltage on the gates as explained in Sec. 2.6. Then we process the data (red) by normalizing our raw measurements and calculating the three statistical channels from those measurements (see Sec. 4.3. After labeling, we end up with input mask and csd (green) to be fed to our ml model (yellow). The outcome of our model is a prediction of the transition lines in the stability diagram. From this prediction mask we perform the necessary postprocessing steps explained in 5.4 and we find the single electron or hole regime (blue). Finally we apply the gate voltages of that regime as we extract them from the prediction mask (gray). 57

5.2 Original U-net architecture with slight modification. We kept the structure and numbers of the original paper [97] we only highlighted how our implementation adapts the U-net architecture by using a pre-trained MobileNetV2 as the encoder and a custom decoder which we describe in Sec. 5.2.2. The encoder and decoder are highlited in red and green respectively. Blue boxes corresponds to feature maps having multiple channels which are denoted on top of each box. The spatial dimensions of the image are presented on the lower left edge of each box. In our case the starting input has a dimension of 1024x1024. The gray arrows are skip connections as highlighted in the legend, and the white boxes are the copied feature maps which gets concatenated during upsampling. (Adapted from [97]) . 58

5.3 Representation of how every csd comes from a die on a wafer 61

5.4 Three charge stability diagrams acquired from the same physical device under different cooldowns and measurement conditions. Despite identical device geometry, variations in thermal noise, contrast, and drift in transition lines are evident between measurements, demonstrating that each diagram represents a distinct sampling of the device's state rather than redundant data. 62

5.5 **Plot showing inference and the main postprocessing steps.** We chose a diagram in which our model's prediction is not very "clean" in order to highlight the importance of the postprocessing steps we implement, which were discussed in Sec. 5.4. (a) The input csd in the form of an image. (b) The ground truth mask labeled as discussed in Sec. 4.2 and used for training the model. (c) The raw model's prediction where each pixel is assigned a probability of how confident the model is that this pixel belongs to a transition line. (d) After thresholding the previous probability map based on a fixed value of 0.75 we get a binary map with the white pixels as lines and the black as background. (e) We apply morphological closing and area dynamic filtering to connect broken lines and remove small spurious detections. (f) On the final cleaned binary mask we find the first two transition lines and locate the center of mass of the single electron regime which we represent here by a red square patch overlaid on top of all the images at the same position. 65

6.1 Examples of successful single charge regime detection gathered from different devices and showcasing different measurement quality. In the first column we have the original csd that is used as input to the model, next to it we have the corresponding hand-labeled ground-truth masks. Th third column is the model's output which consist of a binary mask with the predicted transition lines and the localization of the single charge regime. The single charge region is highlighted by a blue and green contour and its center is indicated by a red square patch. The red patch is overlaid on both the stability diagram and the ground-truth mask to demonstrate accurate localization. 72

6.2 Examples of low-quality stability diagrams from device H, highlighting poorly defined Coulomb peaks, very faint or discontinuous transition lines, and large blank regions with little usable signal. These characteristics are consistent with the device design and measurement quality that differ from the other devices and likely explain the elevated failure rate for device H. 74

6.3 Red arrows pointing to a stochastic line in the input csd (first image) which was labeled as a transition line in the ground truth mask (second image) but was not detected by our model (third image). 78

6.4 Example of a false detection due to a spurious line. From top to bottom we show the input csd, the labeled ground truth mask, and the faulty prediction, with the red arrow pointing at the spurious line which was wrongfully predicted as the first transition by our model. 79

6.5 Accurate detection of the single electron regime in the presence of a spurious line before the first transition (the red arrow is pointing at the spurious line). 80

6.6 Example of a faulty detection due to a fragmented transition line in the prediction mask. 81

7.1 Schematic of a binary image containing the predicted transition lines in white, showing the extraction of possible geometric features which holds relevant physics information. In blue we have the separation distance between transition lines which can be mapped to a voltage value ΔV, and in yellow we show the extraction of the of gate voltage $V_{first\,transition}$ (on the x-axis) corresponding to the first transition line. The numbers denote how many charged particles each region would trap. 87

A.1 Schematic showing the difference between the regular and separable convolution block which is implemented in the MobileNet architecture (Adapted from [98]). 92

A.2 Training and evaluation metrics for the segmentation model. From left-to-right, top row: (a) Loss, (b) Mean IoU. Middle row: (c) Dice coefficient, (d) Pixel Accuracy. Bottom row: (e) Precision, (f) Recall. 95

B.1 Additional successful inference examples (part A). Each panel shows the original stability diagram, the hand-labeled ground-truth mask, and the model prediction (legend as in Figure 6.1). 98

B.2 Additional successful inference examples (part B). 99

List of Tables

2.1 Examples of two-level systems used to realize qubits 12

3.1 Classical charge-tuning methods 39
3.2 ML-based charge-tuning methods 40

4.1 Hierarchical breakdown of CSD counts by mask, polarity, batch, wafer, and design 47

6.1 Per-device design aggregate success of detecting the single charge regime 73
6.2 Per-fold single charge detection success obtained from 5-fold group cross validation. 75
6.3 Summary of common failure modes observed in our stability diagrams and associated causes. 75

A.1 Fixed hyperparameters used for all folds. 91

B.1 **Inference Summary per Design and per Fold.** For each device and fold we report the ratio of the number of diagrams with successful single charge detection to the total number of diagrams, followed by the percentage value. 97

Abbreviations

2DEG Two-Dimensional Electron Gas. 14

2DHG Two-Dimensional Hole Gas. 14

AI Artificial Intelligence. 22, 24, 43

ANN Artificial Neural Network. 22, 26–28

BOX Buried Oxide. 16

CMOS Complementary Metal–Oxide–Semiconductor. 14, 16

CNN Convolutional Neural Network. i, 22, 24, 26, 28, 37, 39–44, 84, 88, 89

CSD Charge Stability Diagram. i, vii, 11, 24–26, 38–43, 46–49, 51–53, 55–57, 61, 64–66, 69, 71, 72, 74–80, 82, 84–86, 89, 96

CV Computer Vision. 22, 24, 28

CWP Cryogenic Wafer Prober. 46

DL Deep Learning. 22, 24, 28

FD-SOI Fully Depleted Silicon on Insulator. 14, 16

FinFET Fin Field Effect Transistor. 14

GPU Graphics Processing Unit. 42, 60, 61, 63, 84

IoU Intersection over Union. 32, 94

ML Machine Learning. i, iii, 9, 10, 22, 24–26, 34, 36, 37, 40, 41, 43, 44, 46, 47, 49, 53–55, 57, 68, 69, 76, 84–86, 88, 89

MOS Metal–Oxide–Semiconductor. 14

MOSFET Metal–Oxide–Semiconductor Field Effect Transistor. 14

NN Neural Network. 28

QD Quantum Dot. i–iii, v, 7–23, 34–40, 43, 44, 46, 50, 66, 67, 75, 85, 87–90

QEC Quantum Error Correction. 4, 6

QPC Quantum Point Contact. 21

SEM Scanning Electron Microscope. 16

SET Single Electron Transistor. vi, 21, 22, 46, 75, 86

SNR Signal to Noise Ratio. 46, 71, 75, 76

Chapter 1

Introduction

The Historical Origins of Quantum Mechanics

The advent of quantum mechanics in the early 20th century constituted a fundamental paradigm shift in physics, arising from the inability of classical theories to describe phenomena at the atomic and subatomic scales. The inception of the ideas that evolved to form what is now known as quantum mechanics can be traced back to the year 1900 when Max Planck hypothesized that energy is emitted and absorbed in discrete packets, or *quanta*, in order to resolve the inconsistencies in the theory of black-body radiation [1]. This concept was further substantiated by Albert Einstein in 1905, who proposed that light itself is quantized into particles called photons to explain the photoelectric effect [2].

The subsequent development of the field led to a more complete, albeit counterintuitive, model of physical reality. Niels Bohr's 1913 model of the atom introduced the quantization of electron energy levels, successfully explaining atomic spectral lines [3]. In 1924, Louis de Broglie extended this concept of quantization by postulating the wave-particle duality of all matter [4].

The mid-1920s witnessed the formulation of a rigorous mathematical framework for quantum theory. Werner Heisenberg's matrix mechanics (1925) and Erwin Schrödinger's wave mechanics (1926) provided two equivalent formalisms for describing the dynamics of quantum systems [5, 6]. The Schrödinger equation, a central tenet of this framework, governs the temporal evolution of a system's wave function, Ψ:

$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t) = \hat{H}\Psi(\mathbf{r},t)$$

The probabilistic nature of quantum phenomena was formalized by Max Born, who interpreted the squared magnitude of the wave function, $|\Psi|^2$, as the probability density of a particle's location. This inherent indeterminism was further solidified by Heisen-

berg's uncertainty principle (1927), which posits a fundamental limit to the precision with which complementary observables, such as position (x) and momentum (p), can be simultaneously known ($\Delta x \Delta p \geq \hbar/2$) [7].

The First and Second Quantum Revolutions

These developments in quantum mechanics enabled us to understand the periodic table, chemical interactions, and electronic wavefunctions, which spurred the **first quantum revolution** that led to the development of technologies like the transistor, the laser, and magnetic resonance imaging (MRI), among others. Transistors in particular hold special importance as they are the main building blocks of digital classical computers, upon which the digital information age was built.

We are currently in the era of the **second quantum revolution**. This phase is distinguished by the ability to precisely control and manipulate individual quantum systems [8]. Advances in experimental physics now permit the isolation, initialization, manipulation, and measurement of single atoms, ions, photons, and electrons. This high degree of control is the critical enabler for the development of quantum computers, secure quantum communication networks, and ultra-sensitive quantum sensors.

Whereas the first revolution leveraged quantum theory to explain existing natural phenomena—revealing why elements behave as they do but offering little means to alter them—the second revolution turns us from observers into architects of the quantum realm. We no longer merely understand the periodic table; we engineer entirely new "artificial atoms" (quantum dots, excitons) with tailor-made optical and electronic properties, unlocking novel capabilities in computation, metrology, and beyond.

In essence, the transition from understanding to engineering marks the boundary between science and technology. The work presented in this thesis—centered on the precise control of individual spin qubits—contributes directly to this field of quantum engineering and puts us right at the heart of the second quantum revolution, helping to transform the counterintuitive rules of quantum mechanics into practical tools for next-generation devices.

The Limits of Classical Computation

The progress of classical computation has been successfully described for over five decades by Moore's Law, an empirical observation stating that the density of transistors on an integrated circuit doubles approximately every two years [9]. This exponential scaling has been the primary driver of the digital age. However, the continuation of this trend is now impeded by fundamental physical limitations. As transistor dimensions approach the

atomic scale, quantum mechanical effects, primarily quantum tunneling, lead to significant current leakage and compromise device reliability. Concurrently, the power density and associated thermal dissipation present challenges to further miniaturization.

Beyond these engineering barriers, a more profound limitation of classical computers exists. There is a class of computational problems for which the required resources (time or memory) scale exponentially with the size of the problem input. Such problems are considered intractable for any classical machine, regardless of its scale or speed. Examples include the accurate simulation of complex quantum systems and the factorization of large integers. The impending cessation of Moore's Law, combined with the existence of these classically intractable problems, creates a compelling imperative to develop novel computing paradigms. Quantum computing has emerged as the most promising candidate to transcend these limitations.

Principles of Quantum Computing and Information Science

Before we explain how a quantum computer works, let us delve into how classical computers function. Classical digital computers represent information in *bits*, each taking one of two definite values, 0 or 1. Bits are physically realized by voltage levels in transistors, charge on capacitors, magnetization in magnetic media, or light pulses in optical fibers. Classical logic gates perform deterministic Boolean operations on bits to execute algorithms. In an ideal, noiseless device, the output is a fully predictable function of the input. However, physical bits are not perfectly reliable, transistor switching errors (from thermal noise and manufacturing variability) and charge leakage introduce nonzero bit-flip probabilities (which is when a bit is flipped from 0 to 1 or from 1 to 0). To mitigate this, classical systems employ error-detection and correction schemes to ensure that logical bits remain faithful over long computations.

In a quantum computer, the fundamental information carrying unit is the *quantum bit* or *qubit*, defined on the computational basis states $|0\rangle$ and $|1\rangle$. Unlike a classical bit, a qubit can exist in an arbitrary linear combination

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where α and β are complex coefficients known as probability amplitudes, constrained by the normalization condition $|\alpha|^2+|\beta|^2 = 1$. Upon measurement, the state $|\psi\rangle$ collapses to one of the two basis states therefore destroying the original superposition. The probability of obtaining $|0\rangle$ is $|\alpha|^2$ and the probability of obtaining $|1\rangle$ is $|\beta|^2$.

Similar to classical computers, quantum computers are also affected by noise which

makes it harder to have a fully reliable system. Therefore, inspired by its classical counterpart, Quantum Error Correction (QEC) was developed in order to mitigate these errors. We will discuss QEC in more details in subsequent sections.

Quantum Advantage

The primary objective in the field of quantum computing is to demonstrate **quantum advantage** (or **quantum supremacy**), a point at which a programmable quantum device can solve a problem of practical interest that is intractable for the most powerful classical supercomputers [10]. The power of quantum computing derives from three uniquely quantum phenomena:

Superposition A qubit can occupy both basis states $|0\rangle$ and $|1\rangle$ simultaneously, allowing an N-qubit register to represent 2^N states. Therefore, adding a single qubit doubles the size of the accessible state space, providing an exponential scaling of the computational state space. In contrast, a classical computer has to double the number of classical bits to double its computing power. This exponential scaling underpins the potential for quantum algorithms to explore many computational paths in parallel and achieve speedups beyond classical means.

Entanglement Qubits can be correlated in such a way that the state of one instantly influences the state of another even when separated by large distances. Entanglement enables nonlocal correlations that are essential for protocols like quantum teleportation and superdense coding, and it also underlies the performance gains in many quantum algorithms.

Measurement The act of measuring a quantum system projects the superposition of states into a definite outcome, which is known as the collapse of the wavefunction. This collapse both enables readout of qubit values providing direct access to quantum information, and supplies intrinsically random outcomes that can be used for secure randomness generation and probabilistic sampling in quantum algorithms.

Together, superposition, entanglement, and measurement constitute the foundational resources of Quantum Information Science, the field dedicated to understanding and leveraging quantum-mechanical principles for the acquisition, processing, and transmission of information.

Beyond the information-theoretic phenomena, practical quantum systems rely on **quantum tunneling** to operate. Quantum tunneling is the phenomenon whereby a particle has a finite probability to traverse a classically forbidden potential barrier, due

to the nonzero amplitude of its wavefunction "leaking" through the barrier. Tunneling a purely quantum effect, and it form the basis of the physical realization of our specific qubit technology.

Problem Classes Demonstrating Quantum Advantage

It is critical to note that quantum computers are not envisioned as universal replacements for classical computers. Rather, they are specialized accelerators designed to tackle a specific subset of computationally hard problems. Moreover, ongoing advances in classical algorithms are steadily narrowing the quantum advantage gap, making many tasks once thought to require quantum speedups increasingly tractable on classical machines.

Here, we present specific classes of problems whose structure maps efficiently onto quantum mechanical principles which make them key areas where quantum computers are expected to provide a significant advantage:

- **Quantum Simulation:** As first proposed by Feynman, quantum computers are naturally suited to simulating other quantum systems [11]. This capability is expected to revolutionize materials science, quantum chemistry, and drug discovery, where the exponential complexity of many-body quantum systems renders them classically intractable [12–14].

- **Cryptography:** In 1994, Peter Shor developed a quantum algorithm capable of factoring large integers into their primes with a complexity that scales polynomially with the input size, an exponential speedup over the best-known classical algorithms [15]. The security of widely used encryption systems, such as RSA [16], is based on the classical intractability of this problem. The field of communication also falls under cryptography with potential applications that leverages quantum entanglement to transmit secure information [17, 18].

- **Search and Optimization:** Grover's algorithm provides a quadratic speedup for searching unstructured databases [19]. While less dramatic than the exponential speedup of Shor's algorithm, this can be applied to a broad range of optimization and search problems.

Quantum Error Correction and the Pursuit of Fault Tolerance

A major obstacle to building large-scale quantum computers is the inherent fragility of quantum states. This fragility stems from the fundamental difference between classical

bits and qubits. A classical bit in a digital computer is represented by a transistor that operates in one of two distinct, stable states, "off" (0) or "on" (1). The transistor acts as a binary switch, and these states are separated by a large energy barrier, making them very robust against noise and interference. Errors, such as a bit spontaneously flipping, are extremely rare.

In contrast, a qubit is an analog object, its state is a continuous superposition of $|0\rangle$ and $|1\rangle$. Qubits are extremely sensitive to environmental interactions, a process known as **decoherence**, which corrupts the quantum information encoded in the amplitudes and relative phases of superposed and entangled states. This susceptibility leads to high error rates in quantum computations.

To overcome this, the theory of QEC was developed [20]. The fundamental principle of QEC is to create redundancy by encoding the state of a single "logical qubit" into a larger number of "physical qubits." However, the overhead required for QEC is substantial and presents a big engineering challenge. Current estimates suggest that creating a single, fully error-corrected logical qubit could require anywhere from 10^3 to 10^4 physical qubits [21]. This high ratio is the primary reason that scalability remains a critical challenge.

The ultimate goal is to build a **fault-tolerant quantum computer**, a machine that can perform arbitrarily long computations by actively correcting errors as they happen. Based on current error rates and QEC overhead, it is widely estimated that a truly fault-tolerant quantum computer will require at least one million physical qubits [22] [23] [21]. To give an idea behind these numbers, we can look at the error rates in qubit technologies and compare it with the error rates of classical computing. Depending on the used technology and the setup, a qubit's error rate is around 10^{-2} to 10^{-3} per gate operation. In contrast, classical computing has error rates of around 10^{-15} per operation on a bit before being error corrected, this rate drops to virtually zero after error correction code has been implemented. The main reason for this low number of errors in classical computing is that they benefit from both advanced fabrication techniques and good error correction code. Given that quantum computers operate on a much narrower error margin, we would need advances in both of these avenues in order to achieve a fault tolerant quantum system.

Different Platforms for Qubit Realization

It is important to note that there is no single "correct" platform to encode quantum information or create qubits. It would most likely be the case that different physical platforms would be used for different applications ranging from quantum computing, communication, sensing, etc. Similar to how classical computing make use of different technologies to manipulate classical bits, depending on the objective. For instance, for logic and processing we use transistors to encode bits as voltages levels (high and low),

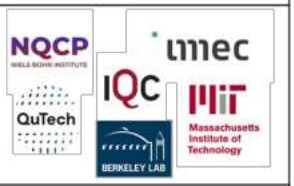
Qubit Type	Pros/Cons	Companies	Research labs
Superconducting	**Pros:** High gate speeds and fidelities. Possibility to use standard lithographic processes. **Cons:** Requires cryogenic cooling, short coherence times; microwave interconnect frequencies still not well understood.		
Trapped Ions	**Pros:** Extremely high gate fidelities and long coherence times. Extreme cryogenic cooling not required. **Cons:** Slow gate times/ Lasers hard to align and scale. Ultra-high vacuum required. Ion charges may restrict scalability.		
Photonics	**Pros:** Extremely fast gate speeds and promising fidelities. No cryogenics or vacuum required. Small overal footprint. Can leverage existing CMOS fabs. **Cons:** Noise from photon loss. Photons don't naturally interact so 2Q gate challenges.		
Neutral atoms	**Pros:** Long coherence times. Atoms are perfect and consistent. Strong connectivity, including more than 2Q. External cryogenics not required. **Cons:** Requires ultra-high vacuums. Laser scaling is challenging.		
Semiconductor Spin/Quantum Dots	**Pros:** Leverages existing CMOS technology. Strong gate fidelities and speeds. **Cons:** Requires Cryogenics. Only a few entangled gates to date. Interference/cross-talk challenges.		

Figure 1.1: Different physical implementations of a qubit: advantages and disadvantages, and most prominent companies and research lab working on each type [29].

and we use capacitors to store short-term data in DRAM memory (charged or discharged), and magnetic tapes for long-term storage, where bits are recorded by locally magnetizing regions of a ferromagnetic material with the orientation of each domain (north up vs down). And data transmission is performed through pulses of current over wires or photons in fiber optics representing bits by the on and off of light pulses.

All of these methods are interchangeable within a single system—data might be fetched from a magnetic disk, held briefly in DRAM, processed in the CPU, and then sent over an Ethernet cable. Each substrate uses the same binary logic (0 and 1) but exploits different physical phenomena—electrostatic charge, voltage levels, magnetism, or photons—to store, manipulate, or transmit bits. Similarly for quantum bits, several physical systems are being actively researched as potential platforms, like superconductive [24], photonic [25], trapped ions [26], and semiconductor QD [27, 28]. We refer to Fig. 1.1 for a non-exhaustive list of the most popular platforms, comparing their advantages and disadvantages.

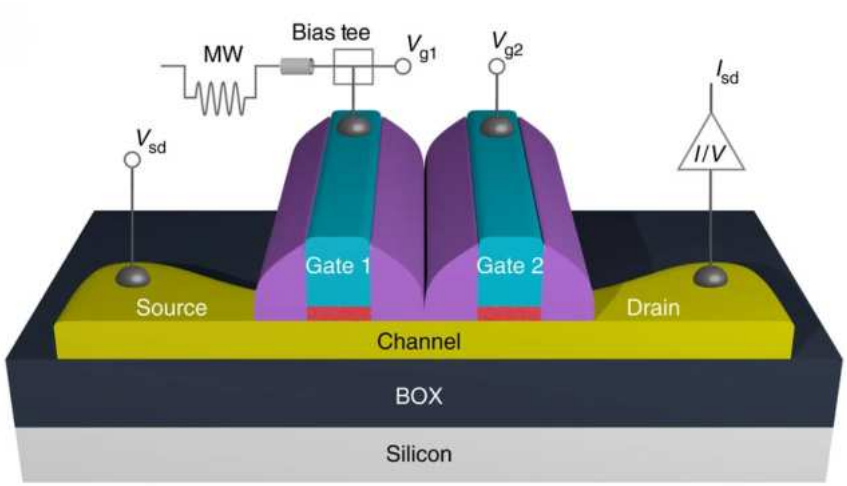


Figure 1.2: Simplified 3D depiction of a silicon-on-insulator nanowire field-effect transistor featuring dual-gate structures. (Adapted from [34])

Silicon Gate-Based Quantum-Dot as Qubits

We will later discuss in detail he different platforms to physically realize qubits, but at this stage we are interested in presenting the motivation behind the specific platform that we are using in this work to build our qubits. In this thesis, we focus on semiconductor QDs [30–33], more specifically we will be working with QDs defined in Si-MOS nanowire similar to the one in Fig. 1.2. These devices offer multiple advantages over the other structures and materials used to realize a qubit, among these advantages we list:

- **300 mm fab compatible:** Leverage mature silicon foundry processes [34–37].
- **High density:** High integration density, which facilitates scalability to larger arrays of qubits. Nanoscale lithography supports large 2D qubit arrays.
- **Long coherence:** We can isotopically purify Si to reach close to zero net nuclear spin, which is normally the main cause of disturbance to the electron spin state, therefore achieving longer electron spin lifetime and better resistance to decoherence. [38–40]
- **Moderate cryogenics:** Silicon-based devices can operate in the range of 1K, which is considerably better than the near absolute zero mK range required for superconductive qubits [41, 42].
- **Short dot–gate separation:** The minimal distance between the QD and gate enables local charge-state readout via gate reflectometry.
- **High gate fidelities** [43–47]

The Tuning Bottleneck

In order to have a qubit-based QD, we need to precisely manipulate the structure to trap a specific number of charged particles in the dot in order to use it as a qubit and encode quantum information. This is referred to as the charge tuning phase and it is one of the many steps that take place during the tuning process (see Sec. 3.1). Charge tuning consist of applying specific voltages on the metallic gates near the dot in order to trap a well-known number of charges inside (e.g., one electron trapped in the QD).

This step is normally done manually because it is hard to automate due to the vast voltage parameter space, which is device dependent and noisy. This is one of the main challenges preventing us from scaling to multi-qubit QD-based quantum computer, because as we scale to multiple qubits (which is required to achieve a useful quantum computer), the parameter space grows exponentially with the size of the QD array and becomes impossible to tune manually. This becomes even more complicated when we consider process variations that are inherent in semiconductor manufacturing which makes it more difficult to tune such device with physics-based models or hard-coded techniques.

The main goal of this thesis is to provide a solution to the charge tuning bottleneck in electrostatically defined QDs that is robust to noise, process variations and can generalize to different device architectures. Our solution is based on ML to automate this process. ML shines in problems that have complex, high-dimensional parameter spaces, noisy and heterogeneous measurements, and no tractable analytical model, these are conditions under which data-driven approaches can learn the underlying nonlinear mapping and robustly generalize across devices..

In this work, we develop a ML-driven charge tuning framework that (1) learns the mapping from gate voltages to charge occupancy, (2) suggests gate voltages for tuning the device to the desired charge occupancy, and (3) adapts to device noise and fabrication variability. By automating charge tuning, we aim to reduce calibration times and lay the groundwork for automated multi-qubit control in large-scale Si-MOS quantum devices.

Thesis Outline

In this introduction, we have laid down the motivation behind quantum computing and how it can solve some of the limitations of classical computing. We also provided the context behind why Si-MOS QD based quantum computers are a good candidate and touched upon one of the main bottlenecks of scaling such a system, which is the tuning process.

In Chapter 2 we build the theoretical foundation of quantum computation, qubit implementations, and semiconductor QDs, their structure, realization, and related physics, along with a theoretical foundation behind ML. Then, in Chapter 3, we formulate the

QD tuning problem, perform a literature review discussing the state of the art methods currently being used, and discuss how ML can be used as a tool for automating this process. We explain the methodology behind acquiring, labeling, and processing the necessary data needed to follow up with our ML model training in Chapter 4. Then, in Chapter 5 we delve into our ML pipeline and architecture, explaining what kind of model we used and the motivation behind our choices. Following that, we present the results of this work in Chapter 6, followed by a discussion of future work in Chapter 9 **??**. Finally, in Chapter 8 we summarize findings, discuss limitations and perspective, and outlines potential extensions to multi-qubit architectures.

Chapter 2

State of the Art

In the previous chapter, we touched briefly on what quantum computing is, the motivation behind its development, and its advantages over classical computers. We also briefly touched on that benefits of using Si-MOS as a qubit realization platform and discussed its scaling issue and the tuning bottleneck. In this chapter, we lay the theoretical foundation behind quantum computing, starting from a dive into the different qubit physical realization, then we focus on QDs, our specific implementation of a qubit. We discuss the physics behind them, quantum confinement, Coulomb blockade, CSDs and the different ways to realize a qubit through QD. Finally, we give a foundational overview of machine learning and its main subtopics.

2.1 Fundamentals of Quantum Computing

2.1.1 The Challenges of Quantum Computers

In 2000, David P.DiVincenzo proposed five necessary conditions for quantum computation [48]:

a. A scalable system with well characterized qubits.

b. The ability to initialize the system in a well defined qubit state.

c. Long coherence time with respect to gate duration.

d. A universal set of quantum gates.

e. The ability to measure individual qubits.

Numerous implementations has been developed that have the potential to satisfy DiVincenzo criteria. In the next section we explain how we can form a qubit structure, then we discuss the different platforms used for qubit implementation.

2.1.2 Physical Platforms to Realize a Qubit

A qubit can be physically realized through any two-level quantum mechanical system where each level would represent one of the two basis states $|0\rangle$ and $|1\rangle$. Some of the most common two-level systems are:

Table 2.1: Examples of two-level systems used to realize qubits

Physical System	$\|0\rangle$	$\|1\rangle$	**Qubit Encoding**
Electron (spin)	$\|\uparrow\rangle$	$\|\downarrow\rangle$	Spin-up vs. spin-down of an electron.
Electron (charge)	$\|0\rangle_{\text{dot}}$	$\|1\rangle_{\text{dot}}$	Empty vs. occupied state of an electron quantum dot.
Photon (polarization)	$\|H\rangle$	$\|V\rangle$	Horizontal vs. vertical polarization of a single photon.
Photon (number)	$\|0\rangle$	$\|1\rangle$	Vacuum vs. single-photon Fock states in an optical mode.
Ion (electronic)	$\|g\rangle$	$\|e\rangle$	Ground vs. metastable excited electronic states of a trapped ion.
Nucleus (spin)	$\|m_I = -\frac{1}{2}\rangle$	$\|m_I = +\frac{1}{2}\rangle$	Two magnetic sublevels of a nuclear spin in a strong field.

2.2 Quantum Dots

Here we will lay the foundation of QDs, what they are, the different ways they can be realized, and how they can be used for qubit implementation.

2.2.1 Quantum Confinement

In a bulk semiconductor, electrons and holes move freely in three dimensions and form continuous energy bands. However, if one or more dimensions of motion are restricted to length scales on the order of the carriers' Fermi wavelength, the spectrum becomes quantized, yielding discrete energy levels. We distinguish three confinement geometries:

- **Quantum well (1D confinement):** Carriers are free to move within a plane but are confined in the perpendicular direction.
- **Quantum wire (2D confinement):** Motion is allowed only along the wire or nanotube axis, and confined in the two transverse directions.
- **Quantum dot (3D confinement):** Complete confinement in all three spatial dimensions. Creates a "particle-in-a-box" system with fully discrete energy levels, analogous to the bound states of an atom.

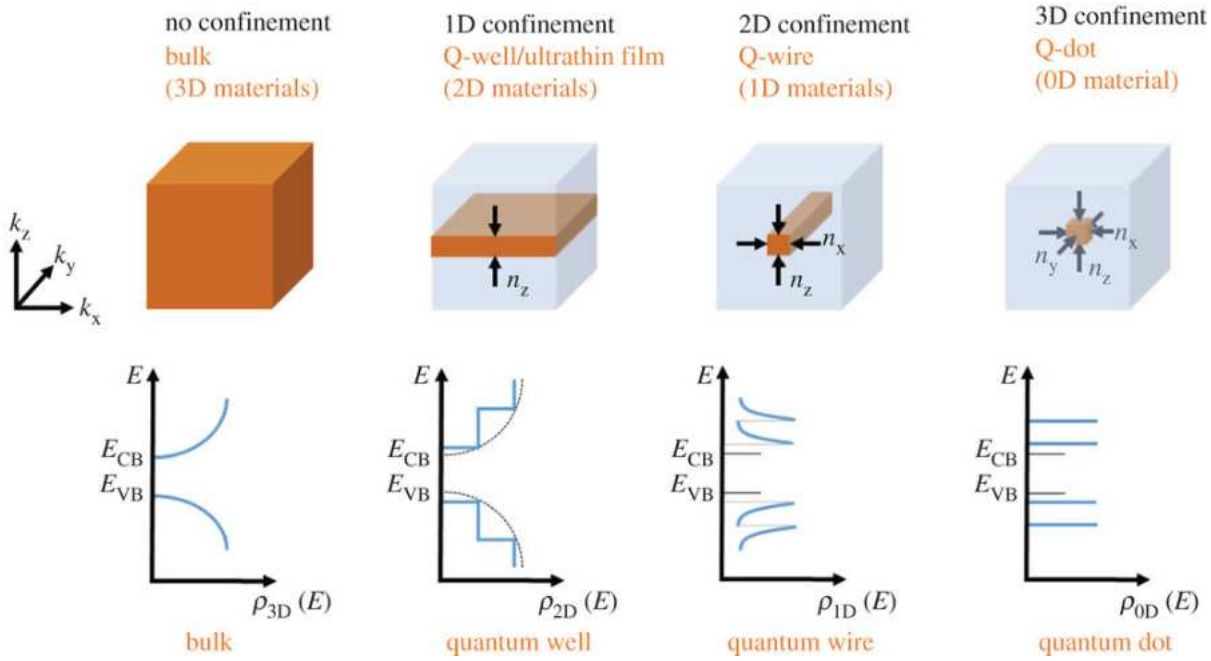


Figure 2.1: Schematic representation of quantum confinement and the discretization of energy spectra as the dimension gets reduced. (Taken from [49])

2.2.2 Quantum Dot Realizations

A *quantum dot* is any structure whose potential confines carriers in all three dimensions, producing discrete energy levels. Two broad approaches exist:

Structural QDs Built directly into the material or heterostructure, where the composition or strain creates a built-in confinement potential:

- **Self-assembled nanocrystals:** InAs islands via Stranski–Krastanov growth [50].
- **Single-molecule junctions:** Molecules bridging electrodes, acting as Coulomb islands [51].
- **Carbon nanotube QDs:** Localized defects or chemical functionalization define 3D confinement along a nanotube axis [52].

Electrostatic QDs Defined by patterned gate electrodes above a or , where applied voltages define potential wells and tunnel barriers:

- **Vertically defined heterostructures:** Layered quantum wells and barriers create dots in the growth direction [53].
- **Laterally defined QDs:** By tuning the gate electrodes we can form barriers and islands to confine charges [54].

In this thesis, we work with *laterally defined* electrostatic QDs formed in a silicon nanowire, where electrostatic gates carve out a single-electron island suitable for spin-qubit operation.

2.2.3 Electrostatically Defined Semiconductor Quantum Dots

There are multiple ways to fabricate laterally defined electrostatic QDs, spanning multiple semiconductor materials and different structures. Listing them all is beyond the scope of this work, therefore in this section we will give a quick overview and then dive deeper into our specific implementation of the silicon nanowire.

Among the different structures to fabricate gate-based semiconductor QDs we list four main approaches:

- Heterostructure
- Metal–Oxide–Semiconductor (MOS)
- Nanowire and Fin Field Effect Transistor (FinFET)
- Dopant donor

We refer to Fig. 2.2 for a representation of these difference device types realized through Silicon. We would also like to note that there are other semiconductor materials being used to fabricate QDs beside silicon, among the most researched are GaAs and Ge which mainly relies on forming QD in the Two-Dimensional Electron Gas (2DEG) or Two-Dimensional Hole Gas (2DHG) formed in a heterostructure of stacked materials.

The specific devices we will be working with throughout our thesis are silicon nanowire Complementary Metal–Oxide–Semiconductor (CMOS) transistors that were fabricated at CEA-Leti clean rooms, on 300 mm Fully Depleted Silicon on Insulator (FD-SOI) wafers. These devices all have the same structure consisting of an undoped silicon channel etched on top of an oxide. The nanowire is connected to a source and drain contacts from both ends. It is also covered with patterned gate electrodes which are used to modulate the electron density within the nanowire. We show a schematic and an electron microscope image of one of our devices in Fig. 2.3a and 2.3b respectively. These devices are different from the typical Metal–Oxide–Semiconductor Field Effect Transistor (MOSFET) in the sense that they do not have the same number and geometry of gates.

The nanowire is a 1D structure, therefore it confines electrons and holes in the two directions perpendicular to it while allowing charges to move freely along the wire. In order to form a QD in the channel we need to confine charged particles along one additional dimension, since QDs are a 0D structures. We achieve this by applying voltages on the gates patterned on top of the nanowire, therefore creating a charge island which is what we call an electrostatic QD.

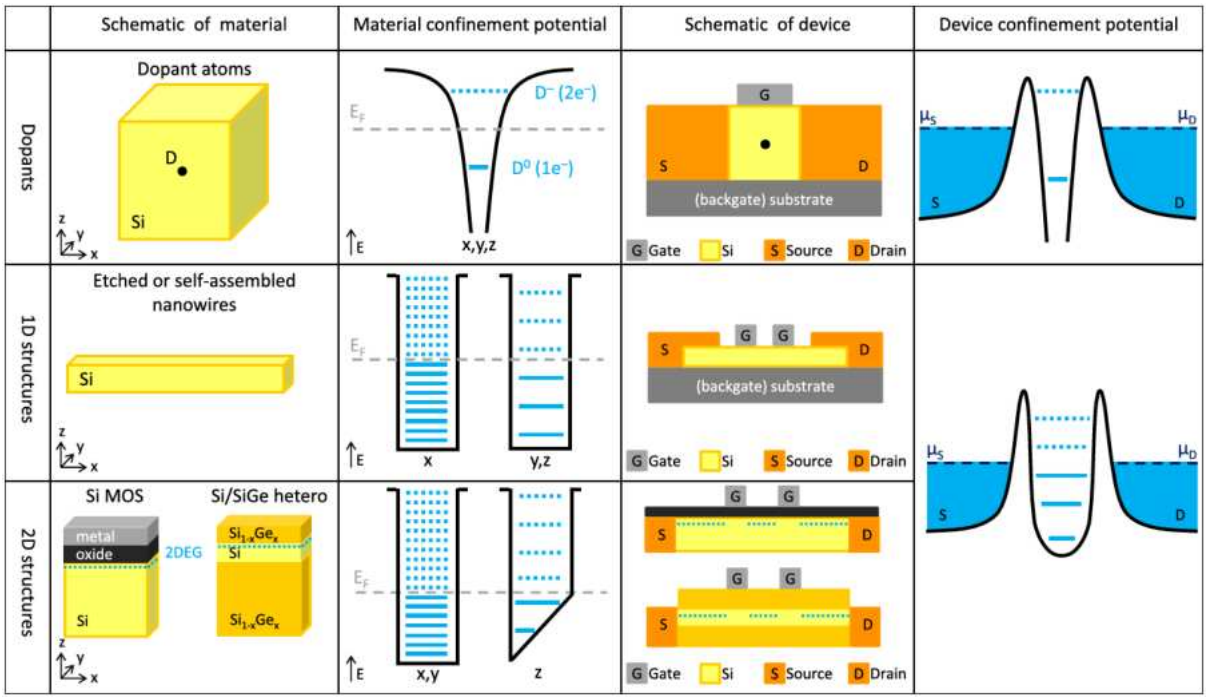
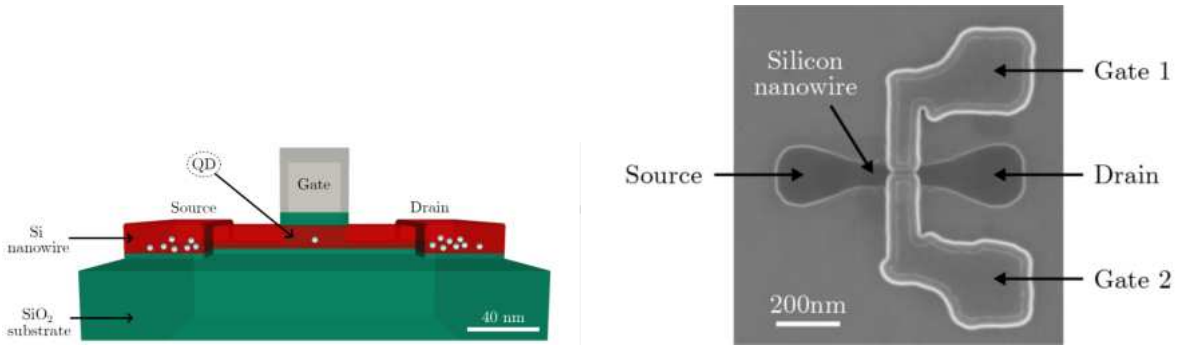


Figure 2.2: Schematic of silicon-based structures for the realization of QDs. First column: Schematic of the structure. Second column: Confinement potential of the electrons in the material. Third column: Schematic of the structure representing the source, drain, and gate electrodes. Fourth column: Potential well electrostatically coupled to gate electrodes that can change the electrochemical potential of the electron relative to the source and drain which are tunnel-coupled to the well. (Reproduced from [55])



(a) (b)

Figure 2.3: **Silicon nanowire CMOS.** (a) Schematic representation of a silicon nanowire (red) on top of an oxide (green) with a gate electrode patterned on the silicon channel (gray). (b) Scanning Electron Microscope (SEM) top-view image of a silicon nanowire CMOS transistor, etched on top of a Buried Oxide (BOX) in a FD-SOI structure. Metallic split-gates are patterned on top of the nanowire which is connected to a source and a drain having the role of electron reservoirs (Taken from [56]).

Different gates have different roles in the manipulation of the QDs and the charge transport between them. Plunger gates are responsible for forming the dots and altering their electrochemical potential. These gates can raise or lower the energy occupation levels of the charge in the potential well of the QD with respect to that of the source and drain. Now there is another type of gates called tunnel barrier which gives us the ability to change the size of the potential barriers between adjacent QDs and also between a dot and the source or the drain. By changing the width of the potential barrier through this gate, we manipulate the probability of having an electron (or hole) tunnel between dots. We can think of plunger gates as giving us the control over how many charged particles we want to trap in each dot, and the tunnel barrier gates letting us manipulate the charge transport between different dots. We visualize this in Fig. 2.4 where we show the potential landscape of a single QD formed by applying a voltage V_G on the plunger gate P, and we show the two barriers B which are manipulated by the barrier gates.

In summary, our gate-based silicon nanowire QD device consists of the following main components:

- **Silicon nanowire:** A one dimensional channel of silicon confining charges along the wire.

- **Plunger Gates:** Metallic electrodes deposited on the surface of the channel which controls the electron potential landscape, forming the QDs.

- **Tunnel Barriers Gates:** Metallic electrodes that can tune the tunneling barrier between dots and allows for coupling adjacent QDs or dot to reservoir.

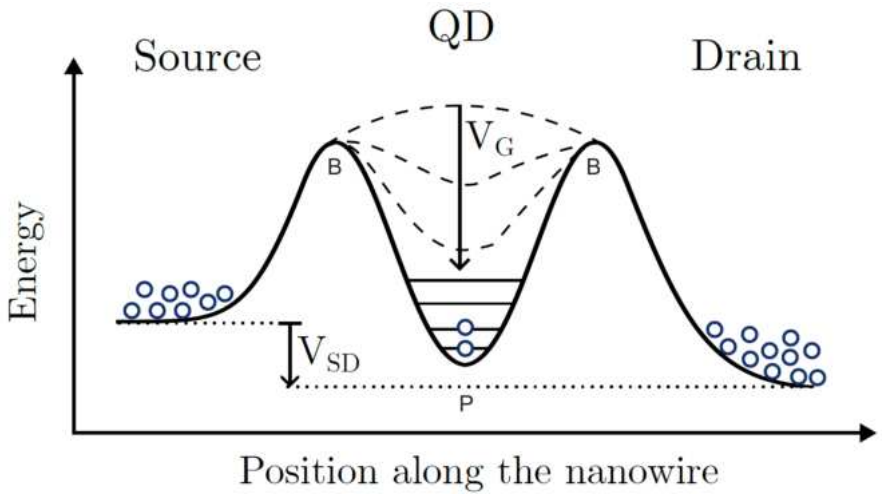


Figure 2.4: Tunable single QD potential profile which is formed by applying a bias voltage V_{SD} between source and drain, and a gate voltage V_G on the plunger gate denoted by P. The QD is formed between two barriers denoted B which can be tuned by changing the voltage on the barrier gates which affects the tunneling rate between the source/drain and the dot.

2.3 Quantum Dots for Qubit Realization

In the preceding chapters, we introduced the concept of a qubit, reviewed the basic elements of quantum computation, and surveyed various physical platforms for qubit implementation. We then examined quantum confinement and explored a variety of quantum-dot (QD) structures and material stacks capable of three dimensional carrier confinement. In this section, we merge these ideas to focus on the use of QDs as hosts for spin-based qubits, which is the central theme of this thesis.

Overview of QD-Based Qubit Modalities

As discussed in section 2.1.2 any two-level quantum system that can be initialized, controlled and readout, can be used to realize a qubit. In essence this means any system that can encode the two basis states $|0\rangle$ and $|1\rangle$. Therefore QDs can be engineered to support a variety of qubit manipulation and encodings. This idea of using QDs for quantum computing was first proposed by Loss and DiVincenzo in 1998 [27]. Below we summarize the most widely studied modalities:

- **Charge Qubit** Two adjacent dots share a single electron. The states $|0\rangle$ and $|1\rangle$ correspond to the electron localized in the left or right dot, respectively.
- **Single-Spin Qubit (Loss-DiVincenzo qubit)** [57] A single electron trapped in a QD has two spin states, up $|\Uparrow\rangle$ and down $|\Downarrow\rangle$, split by a Zeeman energy in an external magnetic field. These states can be used to encode the two logical states of a qubit .

- **Singlet–Triplet Qubit** Two electrons in a double-dot occupy either the singlet $|S\rangle = (|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle)/\sqrt{2}$ or triplet $|T_0\rangle = (|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle)/\sqrt{2}$ state.
- **Triple-Dot (Exchange-Only Qubit)** [58] Three electrons distributed over three dots.
- **Hybrid Qubit** [59] Combines the charge-like fast gating of a double dot with a third electron spin to create a three-level system, where two levels form the qubit.

Regardless of which method we choose it is essential to have control of the charge occupancy of each QD, which means being able to change the gate voltages in order to have a desired number of charges (electrons or holes) in the QDs. Generally speaking the number of charges we would like to trap in each dot will be between 1 to 3, this gives us the ability to manipulate individual charges and encode quantum information through them.

2.4 Coulomb Blockade in QD

We mentioned before that the energy spectrum of a QD becomes discrete because charge carriers are confined to a region whose size is comparable to their Fermi wavelength. This confinement gives rise to quantized orbital levels E_n, with spacing

$$\Delta E \;=\; E_{n+1} - E_n,$$

which grows as the dot shrinks in size.

Charging Energy

Adding an additional electron (or hole) to the QD would cause Coulomb repulsion which is a classical electrostatic effect that take place between charged particles. Therefore the addition of a charge would require extra energy to overcome this repulsion between the charges in the dot and the one we want to add. And this energy cost is referred to as the charging energy E_C.

We can model the QD as one plate of a capacitor (capacitance C), the other plate being an electron reservoir. If V is the potential difference between them, the charge in the dot is

$$Q = C\,V$$

and the electrostatic energy stored in it is:

$$E \;=\; \frac{1}{2}\frac{Q^2}{C} \;=\; \frac{1}{2}\frac{(Ne)^2}{C},$$

where N is the number of electrons already in the dot and e is the elementary charge. The extra energy required to add one more electron (i.e. the charging energy) is the difference

$$E_C \;=\; E(N+1) - E(N) \;=\; \frac{(N+1)^2e^2 - N^2e^2}{2C} \;=\; \frac{2Ne^2 + e^2}{2C} \;\approx\; \frac{e^2}{C}.$$

Thus, to first order the charging energy is

$$\boxed{E_C = \frac{e^2}{C}}.$$

Note that the capacitance C of the QD depends on the spatial extent of the charges' wavefunctions and its coupling to the environment.

Addition Energy

Now the total energy required to add a charged particle to a QD (which is the equivalent of going from N to $N+1$ particles in the dot), is nothing more than the sum of the charging energy of the dot, and the orbital energy spacing $\Delta E \;=\; E_{n+1} - E_n$, between the two quantum energy levels involved in the electronic transition. This energy is referred to as the addition energy:

$$E_{\text{add}} \;=\; E_C \;+\; \Delta E$$

Coulomb Blockade

The charging energy E_C and orbital level spacing ΔE set the scale for electron transport through a QD. If we apply a voltage bias between source and drain

$$V_{\text{SD}} = V_{\text{S}} - V_{\text{D}},$$

a potential difference will be created which allows charge carriers to flow. If an energy level of the dot falls between this created window bias, then an electron will tunnel from the source reservoir to the dot and then to the drain and therefore electron transport is allowed. Otherwise, electron tunneling is suppressed and we have what is known as **Coulomb blockade**. In order to get out of this blockade, we can alter the ladder of electrochemical potentials of the dot in which we can raise it or lower it by changing the voltage of the plunger gate of that dot V_{G}. Therefore by setting an energy level of the dot between the potential of the source and the drain

$$\mu_S \geq \mu_N \geq \mu_D$$

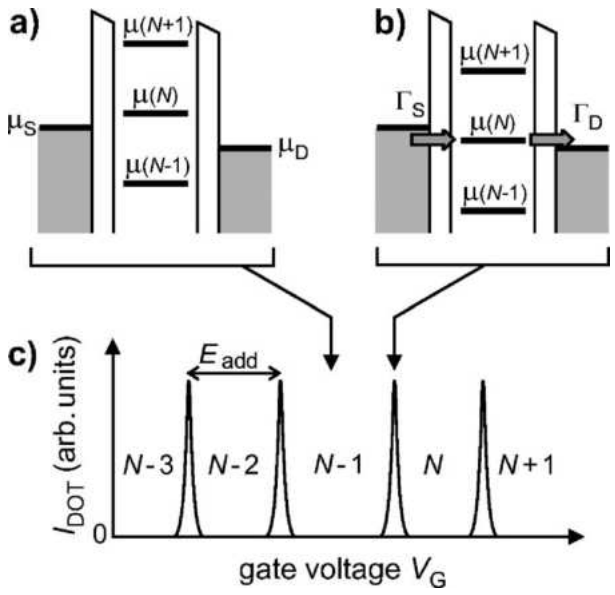


Figure 2.5: Schematics of the electrochemical potential levels of a QD in the low bias regime (a and b), and the resulting one-dimensional trace of Coulomb peaks and blockades (c). For an applied bias voltage V_{SD}, a small window opens to allow charge flow between source and drain (a). By changing the voltage V_{G} of the plunger gate of the dot, we can manipulate the electrochemical potential of that dot, and by positioning an energy level μ_N of the dot between that of the source and drain (μ_S and μ_D respectively), current will flow from source to drain though the dot and therefore single-electron tunneling will take place between $N-1$ and N (b) which can be visualized as Coulomb peaks in the current vs gate voltages plot (c). When this electrochemical energy of the dot μ_N resides out of the bias window, no current will flow from source to drain and the dot is in the Coulomb blockade regime where the number of electrons is fixed at $N-1$. The tunnel rate between the dot and the reservoirs (Γ_S and Γ_D) dictate the magnitude of the current. (Adapted from [28])

we lift the blockade and we allow an electron to tunnel through (see Fig. 2.5). Here $\mu_N = E(N) - E(N-1)$ is the N electrochemical potential of the dot. Once this electron has reached the drain and the dot is empty again, another electron will tunnel through.

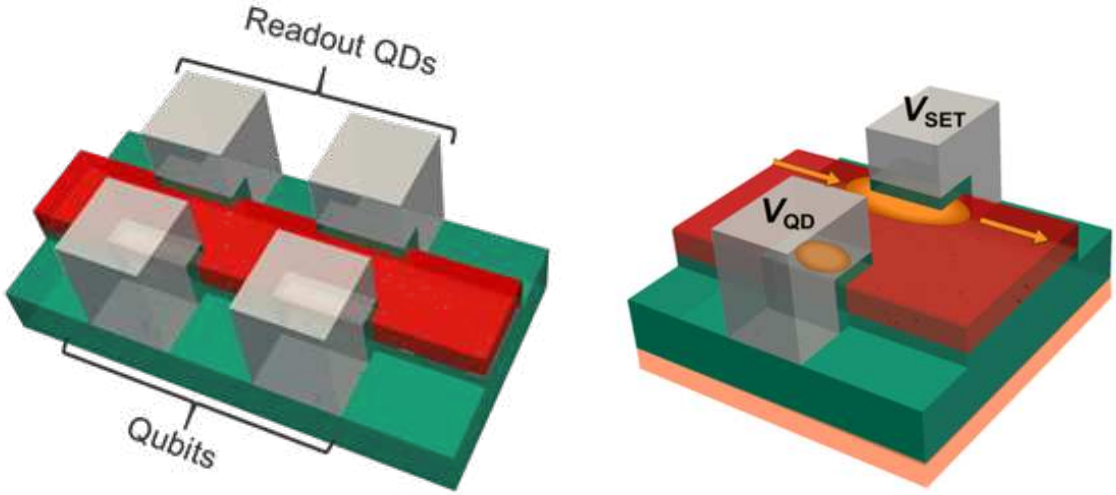
Coulomb Peaks

By sweeping the gate voltage V_{G} and measuring the current flowing through the dot I_{DOT} we get a trace of equally spaced peaks which are known as the **Coulomb peaks**, that are separated by regions of vanishing current (the blockade valleys). These peaks are caused by having a dot energy level fall in the bias window between source and drain, as discussed before. The distance separating these peaks is the addition energy E_{add} and the width of the peaks depends on the applied bias voltage V_{SD}. Also note that the electrochemical potentials of successive energy levels are spaced by the addition energy $E_{\mathrm{add}}(N)$ (Fig. 2.5).

These one-dimensional Coulomb traces form the basis for more sophisticated two-dimensional *charge stability diagrams*, where the current is mapped against two gate voltages. Such diagrams reveal the full regions of charge occupations in single and multiple QDs, and will be discussed in section 2.6.

2.5 Readout Method: Single Electron Transistor

Readout of a QD spin qubit relies on converting the spin or charge state of the dot into a measurable electrical signal. In practice, this is most commonly achieved via a nearby charge sensor, either a SET or a Quantum Point Contact (QPC), whose conductance depends on the occupation of the dot [60–62].



(a) (b)

Figure 2.6: **Readout Method for the Silicon nanowire CMOS.** (a) Schematic representation of a silicon nanowire with the top gates being used as a readout SETs for sensing, and the bottom gates to form few-charge QD to host qubits. (b) Similar device under operation, where we can see the few-charge being formed under the gate with voltage V_{QD} and many-charge SET formed under the gate with bias V_{SET}.

In the devices we use which were fabricated at CEA-Leti, we employ a QD that functions as an SET as our sensing method. The general idea behind an SET charge sensor is to detect a change in signal due to changes in the nearby charge configuration. Our SET consist of a charge island (electrostatically defined QD) that is capacitevly coupled to the QD under test (the one used to form qubits). When a charged particle tunnels into our dot, the electrostatic potential of the SET shifts due to this change of charge configuration, and we detect this shift through current measurement, therefore translating a change in charge occupancy of our target dot into measurable current through our SET.

As we show in Fig. 2.6a we consider one array of gates (bottom) having a role of forming QDs in the few-charge regime which are used to form qubits, and a second array

of gates (top) used to form many-charge QDs which operate as SETs and are used as our sensing mechanism. We can visualize this structure under operation in Fig. 2.6b where we have two parallel gates to form a single QD and an SET facing it, as a readout method.

2.6 Charge Stability Diagram

A charge stability diagram is a 2D current map as a function of gate voltages that gives information about the charge occupancy of each QD. It can tell us precisely what voltage values we need to apply to the gates in order to add or remove a charged particle from a dot. We get such a diagram by sweeping the plunger gates voltages while measuring the current. Our setup for acquiring stability diagrams is similar to the one in Fig. 2.6b. First we configure a QD in the many-electron regime which would act as an SET. This is the readout method we employ as discussed in Sec. 2.5. Another QD is configured in the few-electron regime, down to a single electron. By sweeping the voltages on both gates we get a 2D current map known as the charge stability diagram. In Fig. 2.7 we show an example of a diagram we have measured, in which we can clearly see the Coulomb peaks and the vertical charge transition lines. These transition lines delimit the different charge regimes, therefore by locating them we can identify the charge occupancy of a dot.

2.7 Machine Learning

ML is a subfield of Artificial Intelligence (AI) focused on the development of algorithms that enable computers to learn patterns and make decisions from data without being explicitly programmed [63]. In this way, ML can be seen as a pathway to AI: every ML system is an AI system, but not all AI systems employ ML techniques. Furthermore, Deep Learning (DL) is a subfield of ML that uses multi-layer (i.e. "deep") Artificial Neural Networks (ANNs) to learn increasingly abstract representations of data [64]. By stacking many layers of processing units, DL methods automatically discover features at multiple levels of abstraction, reducing the need for manual feature engineering. As shown in Figure 2.8, all DL techniques lie within the ML circle, reflecting that every DL model is an ML model but with a particular focus on depth and representation learning.

Computer Vision (CV) is the field concerned with enabling machines to interpret and understand visual information from images and video streams [65]. Traditional CV approaches leverage image processing, whereas modern CV increasingly adopts DL, notably CNNs,to learn both low- and high-level visual features directly from pixel data. Broadly, computer vision tasks can be grouped into these main categories:

- **Image Classification**: Assigning a label to an entire image.
- **Object Detection**: Localizing and classifying objects via bounding boxes.

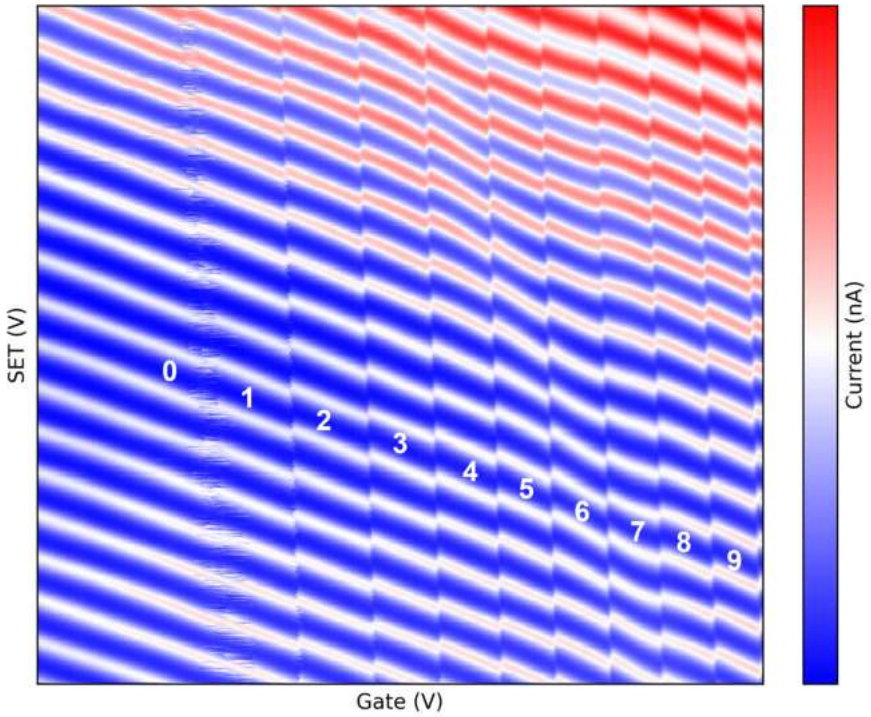


Figure 2.7: **Charge Stability Diagram.** A 2D current map of a QD where the x and y axes represents the voltage sweep applied to the gates, and the color bar represents the measured current value in nA at every point in the map. The numerals represent the number of trapped electrons in the dot, separated by vertical charge transition lines. We can also see the Coulomb oscillations as oblique peaks represented here by the current color bar. The line between the single electron regime and the no charge regime in this case look slightly different from the rest, it is what we refer to as stochastic transition line and it is due to the tunneling rate of the electron between the dot and the reservoir being higher than the sampling rate used for taking the measurement (see Sec. 4.2.1)

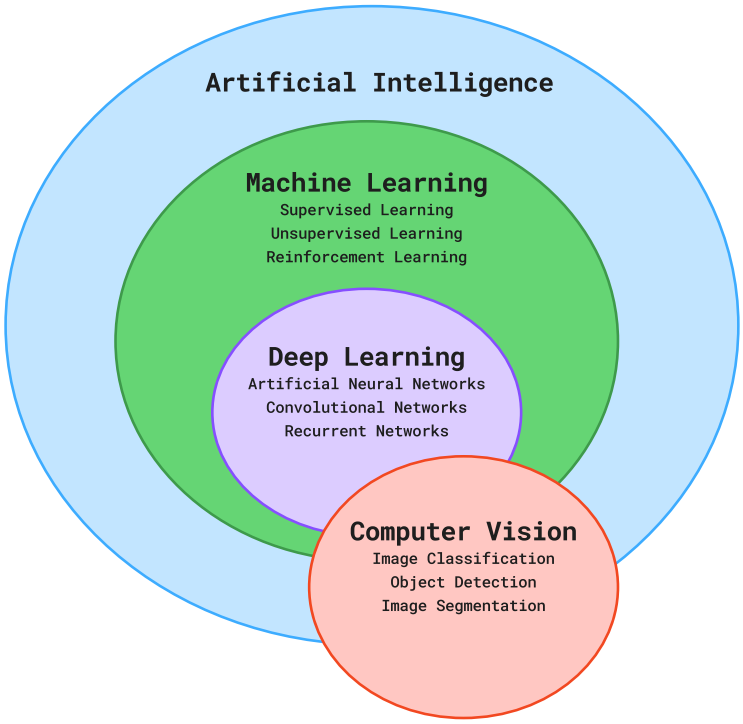


Figure 2.8: Venn diagram showing overlapping relationships between Artificial Intelligence, Machine Learning, Deep Learning, and Computer Vision.

- **Semantic Segmentation**: Assigning a class label to each pixel with predefined categories.
- **Instance Segmentation**: Distinguish individual object instances and their classes at the pixel level.
- **Panoptic Segmentation**: Unified task combining semantic and instance segmentation.

The specific problem addressed in this thesis which is detecting transition lines in stability diagrams, is most naturally viewed as a form of *semantic segmentation*, where each pixel of a CSD image is assigned to either a transition-line or the background. This work resides at the intersection of CV and DL, harnessing the representation power of CNNs for a structured segmentation task within the broader ML and AI framework.

2.7.1 Learning Paradigms

Different paradigms of learning specify how information is presented to the algorithm and what feedback is available during training:

Figure 2.9: The main steps of supervised learning.

- **Supervised Learning:** Models are trained on labeled data to learn a mapping $f: X \to Y$ by minimizing a loss between predicted outputs and true labels [66]. Applications include both regression and classification tasks:
  - *Regression:* Predicting a continuous value.
  - *Classification:* Predicting discrete class labels.
- **Unsupervised Learning:** The model is not provided any labeled data, therefore it does not know what the true answer is and has to find meaningful patterns in the unlabeled data by inferring its own rules. An example of unsupervised learning is clustering.
- **Semi-supervised learning:** combines a small amount of labeled data with a large amount of unlabeled data.
- **Reinforcement Learning:** Agents learn by interacting with an environment, receiving rewards or penalties. [67].

In this work, we use supervised learning on manually labeled transition lines in CSDs (see Sec. 4.2) to predict if a pixel belongs to a transition line or to the background, therefore it is a *binary classification* task. The complete supervised-learning pipeline including data labeling, model construction, training, evaluation, and inference, can be visualized in Fig. 2.9; each step is detailed in subsequent sections.

2.7.2 Data Types

ML models operate on different modalities of data, each requiring specific preprocessing and representation techniques:

Numerical Data: Continuous real-valued features (e.g. pixel intensities, sensor readings) often require scaling or normalization to facilitate optimization.

Categorical Data: Discrete labels or categories (e.g. class labels) must be encoded numerically.

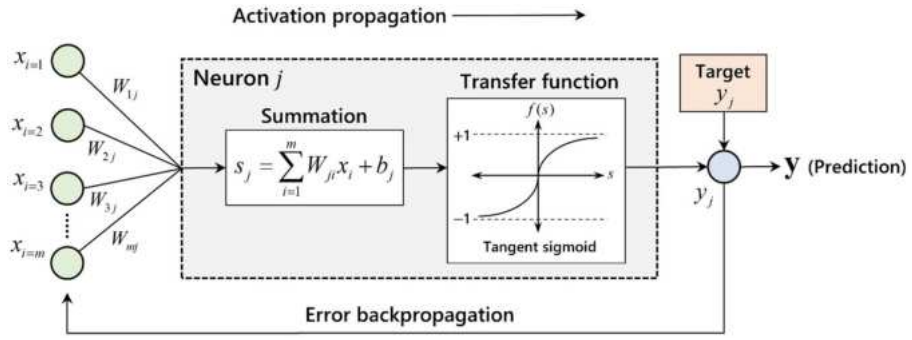


Figure 2.10: Schematic of feedforward and error backpropagation processes for a single neuron j. The neuron performs the computation in Eq. 2.1 and is connected to an input layer which is an aggregation of input neurons x (in green), and a single output neuron y (in blue) which is the prediction of the model (Adapted from [68])

Structured vs. Unstructured: Structured data (tables, time series) contrasts with unstructured data (text, images), influencing model choice and feature extraction pipelines.

In our specific task, we have numerical data which consist of raw current measurements that form our stability diagrams, and we also have categorical data which consist of a label given to the different type of lines that are visible in our CSDs as images (or 2D pixel maps), therefore unstructured data (see Sec. 4.2.1).

2.7.3 What Is a Machine Learning Model

A machine learning model is a parameterized function that maps inputs to outputs. There are different types of ML models, we will only focus on the fundamental ones used in this work. In this section, we introduce ANNs, from single neurons to deep architectures, and then discuss CNNs.

Artificial Neural Networks

ANNs are inspired by the biological brain, composed of interconnected *neurons* organized in layers. Each neuron computes a weighted sum of inputs, adds a bias, and applies an *activation function*, producing an output which is passed to other neurons.

Mathematically, each neuron computes

$$Y = \sigma(\sum_i w_i X_i + b), \tag{2.1}$$

where w_i are the weights, X_i are the inputs, b is the bias, and σ is the activation function. We visualize this step for a single neuron in Fig. 2.10

Activation functions introduce non-linearity into the model, enabling the network to learn complex mappings. Common choices include:

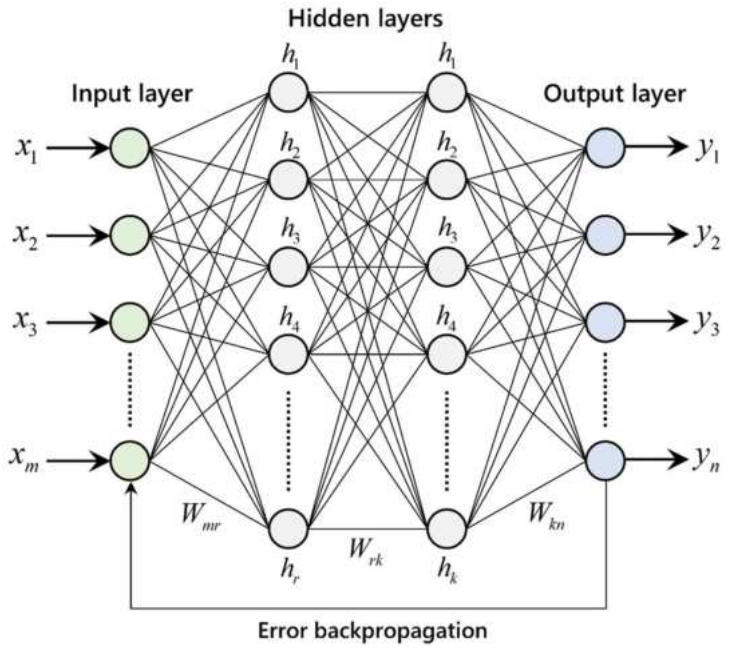


Figure 2.11: Schematic of the architecture of a multilayer ANN with error backpropagation, showing an input layer, two hidden layers, and an output layer. Each node represent a neuron (Adapted from [68])

$$\mathrm{ReLU}(x) = \max(0, x),$$
$$\sigma_{\mathrm{sigmoid}}(x) = \frac{1}{1 + e^{-x}},$$

Networks are typically arranged in an *input layer* (receiving raw features), one or more *hidden layers* (learning intermediate representations), and an *output layer* (producing task-specific predictions); when there are multiple hidden layers, the model is often called a *deep* neural network

At the heart of neural networks is the **feed forward** process, where an input vector is transformed through a series of layers formed by an aggregation of neurons (see Fig. 2.11):

- **Input Layer:** Receives raw data (e.g., pixel intensities from a charge stability diagram).
- **Hidden Layers:** Each hidden layer comprises neurons that perform weighted summations of inputs followed by non-linear activation functions.
- **Output Layer:** Produces the final prediction (e.g., detection of a transition line or classification of charge states).

The depth (number of layers) and width (neurons per layer) of a network controls its capacity to approximate complex functions. Generally, a network that has at least two hidden layers is referred to as deep neural network.

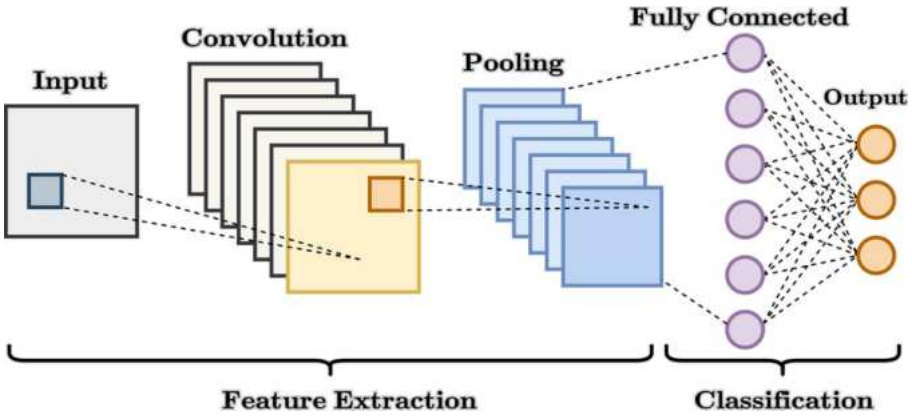


Figure 2.12: A schematic representation of a simple CNN architecture showing the convolution layer, pooling layer, and the fully connected layer (Adapted from [69]).

Convolutional Neural Networks

There are multiple types of Neural Network (NN). Here we will focus on CNN which form the backbone of DL-based approaches to CV (not considering newer architectures like transformers). CNNs extend ANNs by exploiting the spatial structure of grid-organized data (like images). A *convolutional layer* applies a set of small, learnable filters (kernels) across the input to produce feature maps that detect local patterns. The kernel slides over the input map and perform dot product operation (convolution) mathematically defined as such:

$$F(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) \cdot K(i-m, j-n),$$

where I represents the input image, K is the filter or kernel, F is the resulting feature map, and the tuple (i,j) represent the pixel coordinates in a 2D map.

Following convolution, an activation function introduces nonlinearity. A *pooling layer* then downsamples each feature map by aggregating values over non-overlapping windows. Pooling reduces spatial resolution, lowers computational cost, and imparts translation invariance.

Stacking multiple such blocks yields deep convolutional architectures capable of capturing hierarchical patterns—from edges and textures in early layers to complex, task-specific structures in deeper ones.

2.7.4 Training Process of a Neural Network

Training a neural network consists of iteratively tuning its parameters θ so that the model's predictions $\hat{Y} = f_\theta(X)$ align with the true labels Y. At each step, a **forward pass** computes $\hat{Y}$ from a batch of inputs X, and evaluates the **loss** $\mathcal{L}(\hat{Y}, Y)$, a scalar measure of prediction error, i.e. how far is the prediction from the actual true value.

Next, **backpropagation** takes place, in which a backward pass efficiently computes the gradient $\nabla_\theta \mathcal{L}$ of the loss with respect to each parameter [70]. These gradients indicate the direction in parameter space that most rapidly decreases the loss. An **optimizer**, such as stochastic gradient descent (SGD) or Adam [71], then updates θ by taking a small step along this direction; the **learning rate** controls the size of this step and can be held constant or adjusted over time via a scheduling strategy.

A major challenge in training is **overfitting**, wherein the model learns spurious patterns specific to the training data and fails to generalize to new inputs. **Regularization** methods help mitigate overfitting: for example, *weight decay* adds a penalty proportional to the squared magnitude of parameters [72], encouraging simpler models, and *dropout* randomly deactivates a fraction of neurons during each update, preventing co-adaptation of features [73].

To further improve robustness, **data augmentation** applies random transformations, such as rotations, flips, or noise injection to the input data, effectively enlarging the training set and exposing the model to a wider variety of examples [74]. Throughout training, one typically monitors both training and validation losses: if validation error starts increasing while training error continues to decrease, this signals overfitting, and one may invoke *early stopping* to halt training before performance degrades on unseen data.

Together, these steps—forward pass, backward pass, parameter update, and the use of regularization and augmentation—form the core workflow that enables neural networks to learn effective, generalizable representations from data.

2.7.5 Transfer Learning and Fine-Tuning

Transfer learning is the act of **fine tuning** a **pre-trained model** on specific domain data [75, 76]. A pre-trained model is a network whose been trained on a large, general dataset, therefore its parameters has already been optimized and has learned rich, reusable feature representation that capture low-level patterns (e.g. edges, textures) and higher-level abstractions (e.g. object parts, semantic relations). Therefore one can use such pre-trained models to transfer their learning to adapt these representations to new tasks or domains. This is often useful when labeled data is scarce.

Two common strategies are generally used to achieve transfer learning: *feature extraction* and *fine-tuning*. In feature extraction, one freezes the pre-trained layers and trains only a new, task-specific head (e.g. a classifier or regressor), thus treating the frozen network as a fixed feature extractor. This reduces the number of trainable parameters and the risk of overfitting.

In fine-tuning, one allows some or all of the pre-trained weights to continue updating on the new data, typically using a smaller learning rate than for a randomly initialized

network. By selectively unfreezing higher layers (where task-specific concepts tend to reside), one retains general features learned earlier while adapting the model's semantic understanding to the target domain.

Overall, pre-trained models and transfer learning enable rapid convergence, lower data requirements, and often superior performance on downstream tasks, making them essential tools in modern machine-learning workflows.

We employ transfer learning in this work, therefore our segmentation models benefit from robust low-level feature representations while focusing training effort on domain-specific transition-line characteristics.

2.7.6 Evaluation

Since our objective in this work is to accurately identify pixels belonging to transition lines (see Chapters 4 and 5), this problem falls under semantic segmentation, and therefore we focus here on classification-based evaluation metrics.

To assess segmentation quality, we compute both per-pixel classification metrics and overlap measures specific to mask prediction. Let

$$\begin{aligned}
&\text{TP} = \text{True Positive} = \text{pixels correctly predicted as line},\\
&\text{TN} = \text{True Negative} = \text{pixels correctly predicted as background},\\
&\text{FP} = \text{False Positive} = \text{pixels incorrectly predicted as line},\\
&\text{FN} = \text{False Negative} = \text{pixels incorrectly predicted as background}.
\end{aligned}$$

Pixel Accuracy Overall accuracy measures the fraction of all pixels that are correctly classified (either line or background):

$$\text{Accuracy} = \frac{\text{Correct classifications}}{\text{Correct classifications} + \text{Incorrect classifications}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}.$$

This reflects the model's ability to capture both transition boundaries and background regions across the entire stability diagram. However, since transition-line pixels are often $< 1\%$ of the image, therefore, the accuracy can be misleadingly high if the model simply predicts background everywhere. In such a case, a model predicting all pixels as background would have an accuracy of 99% giving a false sense of accuracy.

Precision and Recall *Precision* and *recall* measure correctness and completeness of positive predictions,

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

Precision gives the fraction of predicted line pixels that actually correspond to true transitions. High precision means the model's detected lines are reliable and not confused

by spurious noise.

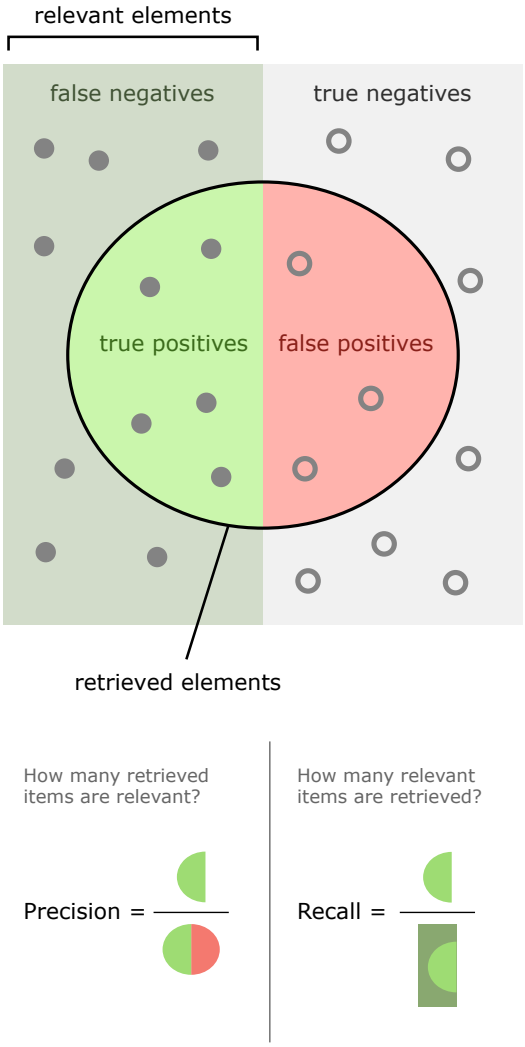


Figure 2.13: Visual representation of Precision and Recall (Adapted from [77]).

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

Recall, also called *sensitivity*, quantifies the fraction of true transition-line pixels that the model successfully detects. In concrete terms, recall measures how many of the annotated transition edges the model recovers.

Dice Similarity Coefficient (DSC) or F_1 Score The Sørensen–Dice coefficient measures the similarity between the predicted mask P and ground truth G:

$$\text{Dice} = \frac{2\,|P \cap G|}{|P| + |G|} = \frac{2\,\text{TP}}{2\,\text{TP} + \text{FP} + \text{FN}}$$

It is the harmonic mean of precision and recall, and penalizes cases where either

precision or recall is low, ensuring both the correctness and completeness of detected transition lines, and is a good metric for class imbalanced datasets.

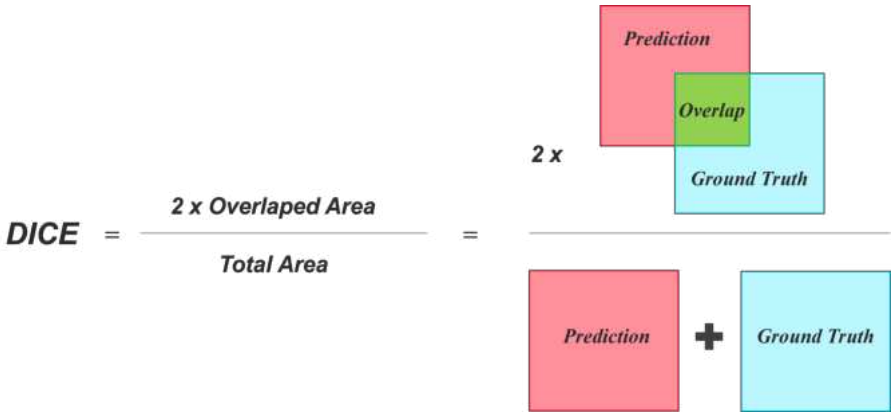


Figure 2.14: Schematic representation of the Dice metric.

$$F_1 = 2\,\frac{\text{Precision}\ \times\ \text{Recall}}{\text{Precision} + \text{Recall}} = \text{Dice}.$$

Intersection over Union (IoU) Also known as the Jaccard index, IoU evaluates pixel-level overlap between the predicted mask P and ground truth G:

$$\text{IoU} = \frac{|P \cap G|}{|P \cup G|} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}$$

IoU represents the fraction of correctly identified transition area relative to the total area covered by either prediction or annotation, penalizing both missed and extra pixels i.e. the IoU penalizes under and over segmentation more that the Dice coefficient.

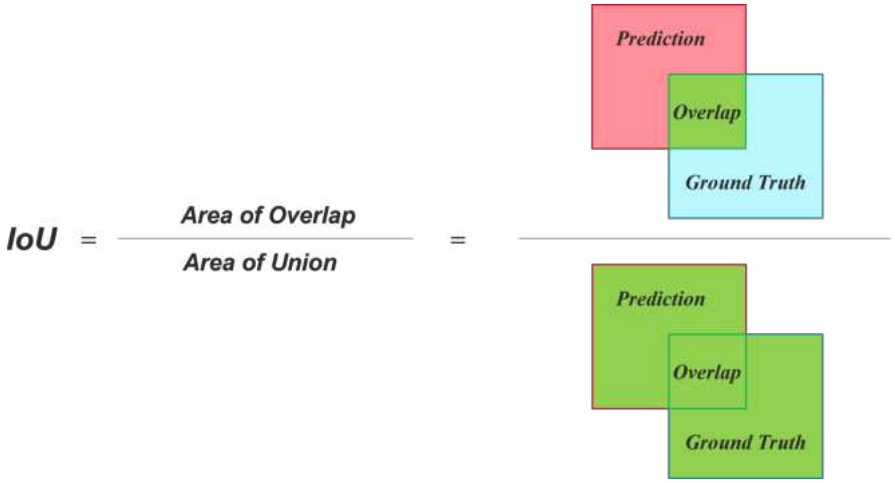


Figure 2.15: Schematic diagram of the IoU metric.

By reporting Accuracy, Precision, Recall, F_1, IoU, and Dice, we provide a comprehensive view of our model's performance, from overall pixel correctness to the precise delineation

of transition-line boundaries. The confusion matrix entries $(\mathrm{TP}, \mathrm{TN}, \mathrm{FP}, \mathrm{FN})$ underpin all these metrics and offer additional insight into common error modes.

2.7.7 Model Inference

Model inference refers to the process of using a trained network to generate predictions on new, unseen data. Unlike training, which iteratively adjusts parameters, inference performs a single forward pass through the network: given an input x, the model produces an output $\hat{Y} = f_\theta(X)$ using fixed parameters θ. Efficiency considerations include model size, latency, and hardware constraints.

2.8 Conclusion

In this chapter, we have established the quantum-mechanical and machine-learning principles underpinning our work, including qubit realizations, quantum-dot physics, Coulomb blockade, charge stability diagrams, and the fundamentals of supervised segmentation. In the next chapter (Chapter 3), we delve deep into the tuning process, perform literature review and present our proposed solution to the problem.

Chapter 3

The Tuning Process of Quantum Dots

We discussed in the previous chapter how QDs can be used for qubit realization. Now this is not a trivial task. We first need to form the QD in a semiconductor heterojunction or nanowire by applying a set of voltages to different metallic gates patterned on the junction. Then we need to trap a single charged particle in the QD in order to manipulate it and perform operations on it as a qubit. This process is known as the tuning of QDs, and it consists of multiple steps which we will detail in Sec. 3.1.

The tuning process is one of the critical challenges that prevents us from scaling a QD based quantum computer. The reason is that as we increase the number of dots in a system, the parameter space for all the possible voltages to be applied to the multiple gates in order to realize a multi-qubit system becomes very large, to the point where heuristic manual tuning becomes infeasible. This necessitates the need to automate the tuning process, and this is what we will discuss in this chapter.

We first start with discussing the multiple steps required to tune a QD, then we highlight the need to automate this process and the motivation behind it, and why it is necessary for scaling. Then we delve into a literature review study discussing the state-of-the-art methods and techniques currently being employed to solve this problem, and compare the different approaches to our methodology. We also present the necessary arguments that motivate the use of ML for such a problem, discussing its advantages over classical techniques.

3.1 Tuning of QDs

Tuning quantum dots involves multiple stages [78], we give a brief overview of them below:

1. **Bootstrapping (Initialization and Measurement Setup):** Establishing initial operating conditions by cooling down the device and bringing the charge sensors into an operational regime.

2. **Coarse Tuning (Device Topology):** Reaching a well-known global configuration of charge islands (single QD or double QD, etc.) with well-defined connectivity by tunnel-coupling the dots. This is achieved by changing the voltages on the plunger gates [79–86]

3. **Controllability:** Ensuring that gate voltages have predictable effects on the QD potential or the tunnel barrier between dots, by implementing virtual gates to compensate for the capacitive crosstalk between the dots [79, 81, 87, 88].

4. **Charge Tuning (Charge Occupancy):** Bringing the device to the desired charge configuration by defining the number of charges in each QD [79, 89–95].

5. **Fine Tuning:** Achieving precise control over the qubit's state for reliable operation [81, 96] .

Traditionally, this tuning process has been performed manually by experts relying on their intuition and informed guesses, which is a time-consuming and error-prone procedure that cannot be easily scaled to large arrays of QDs due to its labor-intensive nature. Given the need for multiple qubits to achieve quantum supremacy, we need to automate the whole tuning process in a way that is robust to fabrication variability which introduces defects in the device, as well as environmental noise like temperature fluctuations, and electromagnetic interference etc.

In this work, we are mainly concerned with the charge tuning step, in which we aim to develop automated and robust methods to isolate a single charge in a QD to achieve the so-called single electron regime. Therefore, in the remainder of this thesis we focus on the step of charge tuning starting by a deeper dive into the process.

3.2 Charge Tuning of QDs

The step of charge tuning comes right after the bootstrapping and coarse-tuning phases. At this stage, we have a well-defined device topology of QDs and an operational charge sensor. But we do not yet control how many charged particles are in each dot, so at this point we have an unknown number of particles in our QDs. The main objective of charge tuning is to achieve a particular charge configuration of our system by trapping the desired amount of charges in each QD. The adopted strategy to achive that is by (i) emptying the QDs of all electrons and then (ii) reloading the desired number of electrons in each dot. This procedure is the only way we could know the charge occupancy, since there is no absolute charge detector. Therefore the only way to count the number of charges is through this kind of measurement which necessitate the unloading and reloading of charges.

This step is very important, because the number of particles we want to trap into each QD depends on the type of qubit we want to achieve (see section 2.3). For most cases, that number will vary from one to three particles per dot. Therefore, we need to have the ability to precisely manipulate the loading and unloading of a single particle at a time into our QD.

In order to trap only one electron in each QD, experts manually change the voltages on the gates after looking at the measured charge stability diagram to locate the single electron regime. This heuristic approach to charge tuning becomes increasingly difficult as the number of dots scale and it becomes a real hurdle to overcome if we want to build a fault-tolerant QD gate-based quantum computer.

One way researchers are trying to overcome this challenge is by completely automating the charge tuning process. Some approaches have been suggested using both classical script-based techniques, and others are relying on ML methods. In this section we lay down the motivation behind using ML techniques for this task. Then in Sec. 3.3 we perform a literature study of the cutting edge techniques being implemented for the charge auto-tuning, discussing both classical and ML-based approaches.

Motivation for Using a Machine Learning Approach for QD Auto-Tuning

The manual charge-tuning process described above suffers from several key limitations when scaled to larger quantum dot arrays. First, it relies on expert operators to visually inspect charge stability diagrams and heuristically adjust gate voltages—an approach that is both time-consuming and susceptible to human error. Finally, as we push toward multi-qubit architectures, the search space of gate-voltage configurations grows exponentially, quickly making manual tuning infeasible.

Therefore, a lot of effort has been devoted to automating the multiple steps of the tuning process over the past years. This started initially by implementing script-based approaches that would follow a programmed logic that would attempt to replicate the steps that an expert would use during tuning. But it has become clear to researchers that these hard-coded methods would not scale well, given the number of different gate geometries used by different groups, nor would these methods be robust to noise or device variability due to fabrication.

This led to shifting the focus towards more robust techniques like ML, that would overcome these hurdles and make it possible to have a general automated approach for tuning that would work on any device, regardless of material stack, gate geometry, or fabrication variability.

Advantages of Machine Learning

Machine Learning provides a natural pathway to overcome these challenges. A well trained ML model with an acceptable amount of good quality data can learn to identify transition lines, infer the correct gate-voltage adjustments, and predict the single-electron regime without human intervention. In particular, CNNs excel at extracting spatial features from images, making them well-suited for the problem of interpreting charge stability diagrams. Other methods using Reinforcement Learning (RL) can also be used for such tasks. Since our approach mostly rely on CNNs we will mostly focus on its benefits.

Here are some advantages of an ML-based auto-tuning system:

- **Adaptability to Variability:** Because ML models learn directly from data including noise patterns, fabrication imperfections, and measurement artifacts, they can generalize across different devices and measurement setups and be robust to variability. This reduces the need for device-specific heuristics and allows a single trained model to be deployed on different quantum dot structures.
- **Robustness to Noise:** By training on data augmented with realistic sensor noise or real measurement data, the model learns to distinguish actual features from spurious fluctuations. As a result, it maintains high tuning accuracy even under low-signal-to-noise conditions and varying experimental environments.
- **Transfer Learning:** Pretrained networks can be fine-tuned on a small set of measurements from a new device geometry or material system, dramatically reducing the amount of labeled data required for reliable tuning. This enables rapid adaptation to novel QD architectures with minimal additional calibration effort.
- **Scalability:** As the number of QDs grows, the tuning problem becomes a high-dimensional optimization task. ML algorithms can handle these high dimensions by identifying correlations across multiple gate voltages and charge transitions, effectively navigating a large parameter space more efficiently than manual search.
- **Consistency and Repeatability:** Automated methods remove the subjectivity inherent to manual tuning. For identical input data, the ML model will consistently recommend the same voltage adjustments, ensuring reproducible single-electron regimes across multiple cool-downs and devices.
- **Continuous Improvement:** The training dataset for an ML model can be augmented over time with new measurements, allowing the model to refine its predictions as more device types and noise conditions are encountered. This continual learning ensures that tuning performance improves as the quantum hardware evolves.
- **Speed and Throughput:** Once trained, a CNN can analyze a new charge stability diagram and output gate-voltage corrections in milliseconds, compared to several

minutes (or more) for a human operator. This rapid inference allows large arrays of QDs to be tuned in parallel, dramatically reducing calibration time.

3.3 Literature Review and State-of-the-Art

Automated charge tuning of QDs aims to set devices into a predefined charge configuration without human supervision. As we mentioned before, all currently known approaches implement a two-phase strategy: (1) *emptying* the QDs of all charges and (2) *reloading* the desired number of electrons, typically monitored by the appearance or disappearance of Coulomb transition lines in CSDs. We categorize existing work into two families: **classical heuristic** methods, and **machine-learning** methods, highlighting their results and key strengths and limitations.

3.3.1 Classical Heuristic Methods

There have been several early attempts to automate charge tuning using purely script-based image-processing, and physics-informed strategies. **Baart *et al.*** were among the first to attempt to transfer the expert's tuning workflow into a machine. They performed classical image-pattern matching on double-QD arrays in GaAs to locate the bottom-left charge crossing, then they checked that no other transition line exist for more negative gate voltages with respect to the most bottom-left detected crossing, and therefore locating the single electron regime [89]. They reliably reached the (1,1) configuration in every tested device . Although achieving 100% success on three devices, Baart's algorithm depends on prior knowledge of gate geometry and pinch-off voltages, limiting its generality to new architectures.

Lapointe-Major *et al.* presented a distinct approach for single quantum dot auto-tuning. Instead of a single full scan, their algorithm implemented an adaptive measurement sequence of subsized stability diagrams. The flow involved an iterative loop: measure a small CSD, process the signal to remove background noise, and then use image analysis (Hough transform or EDLines algorithm) to detect and reconstruct charge transition lines. The algorithm then heuristically determined subsequent measurement regions to systematically deplete the dot of electrons until no more transitions were found, followed by reloading a single electron by moving back across the first detected transition line. The method was experimentally tested against two different silicon-based quantum dot devices, demonstrating good success [90].

To reduce the burden of full 2D scans, physics-informed methods were developed by **Ziegler *et al.***, replacing full images with Physics Informed Tuning (PIT) of 1D "ray" cuts along calibrated virtual-gate axes, unloading and reloading electrons in double QDs [79]. This ray-based tuning module uses a series of 1D measurements and custom peak

finding techniques to find the desired charge configuration. Tested on both simulated and offline experimental GaAs and Si data, this method delivered 89.7 % success rate on experimental data and 95.5 % on simulated.

Table 3.1: Classical charge-tuning methods

Reference	Method	QD Type	Data	Success
Baart *et al.* (2016) [89]	Image pattern matching	Double QD	Experimental	100%
Lapointe-Major (2020) [90]	Signal and Image processing (Hough transform, EDLines)	Single QD	Experimental	100%
Ziegler *et al.* (2023) [79]	Physics-informed rays	Double QD	Experimental/Simulation	89.7–95.5%

3.3.2 Machine Learning Methods

As these classical approaches reached their limits, either in requiring device-specific knowledge which makes them not easily generalizable, or in handling noisy, variable data, machine learning emerged as a promising alternative.

Durrer *et al.* for instance applied a two-stage convolutional neural network to GaAs double QDs: a low-resolution CNN for unload scans and a high-resolution CNN for reload scans [91]. Despite training on $\sim 10^5$ augmented experimental images, their overall tuning success remained only ∼57% (90% unload, 63% reload), underscoring the fragility of local patch inference under realistic noise. No information was given regarding the time taken by the algorithm to find a desired charge configuration.

Czischek *et al.* uses small feed-forward networks trained on synthetic patches of CSDs to detect transition lines in single QDs [92]. By stepping through a fixed patch grid, they achieved 98% success in emptying but only 53% in reloading, later improved to 75% with a 4×4 patch array. These patch-based methods minimize measurement area but can be derailed by "gaps" in sparse transitions that the fixed exploration path skips. Their method was tested on 27 CSDs that were measured from 3 devices.

Yon *et al.* then introduced Bayesian CNNs with uncertainty quantification, trained on experimental CSDs of Si and GaAs based single-QD devices. The problem is framed as an exploration task, where each step involves detecting charge transition lines in small subsections of the voltage space referred to as a patch, aiming to reach a specific charge regime while minimizing measurement time. Their model was trained on patches extracted from the CSDs of 3 different single-QD devices consisting of 17, 9, and 12 diagram per device. They achieved an offline tuning success of of 99.5% (Si-SG), 80.6% (GaAs), and 78.1% (Si-OG) [93], which was tested on 9 CSDs for each device. While this approach help minimize the measurement time and does provide a larger number of images for the model to train on (due to splitting each diagram into many patches), it

would be difficult to differentiate between a regular transition and a spurious line due to the intrinsic nature of the patch approach which does not take into account the whole diagram.

Their follow-up work demonstrated 95% in online auto-tuning of a single overlapping-gate Si QD which was performed over 20 runs [94] with the only failed attempt at auto-tuning being due to a problem in the exploration logic and not the CNN classification, providing further argument against hard-coded logic and more focus on ML-based approaches. The average tuning run took 2 hours and 9 minutes, with 96% of this time consumed by measurement, and only 4% for data transfer, processing, and inference. This work is an impressive achievement which they attributed to the good fabrication quality of the device they used for the test. It could be interesting to see the results on a bigger variety of devices to benchmark because the 20 runs tested in this experiment were performed on the same device which means the model was seeing different parts of the same stability diagram therefore it could be interesting to test it against different type of noises and irregularities.

Most recently, **Schuff *et al.*** integrated Bayesian optimization, CNN feature detection, and computer vision into a fully autonomous spin-qubit tuning pipeline for Ge/Si nanowire QDs. Over 13 runs, 77% reached single-spin Rabi oscillations; total tuning took 22–80h/run (avg. 38h), also dominated by measurement time [95].

Table 3.2: ML-based charge-tuning methods

Reference	**Method**	**QD Type**	**Data Type**	**Test Size**	**Test Type**	**Success**	**Online Tuning Time**
Durrer *et al.* (2020) [91]	CNN (two-stage)	Double QD	Experimental	160 tuning runs across 2 devices	Online tuning	56.9%	—
Czischek *et al.* (2022) [92]	FFNN on patches	Single QD	Simulation/ Experimental	27 CSDs from 3 devices	Offline tuning	98% empty 75% reload	—
Yon *et al.* (2024) [93]	Bayesian CNN - patch based	Single QD	Experimental	27 CSDs across 3 device types	Offline tuning	78.1% - 80.6% - 99.5%	—
Yon *et al.* (2025) [94]	CNN	Single QD	Experimental	20 runs on a single device	Online tuning	95%	2.15h
Schuff *et al.* (2024) [95]	Deep Learning, Bayesian optimization, Computer Vision	Double QD	Experimental	13 runs	Online tuning	77%	22–80h total tuning

3.4 Proposed Full-Diagram Charge Tuning Method

Our approach for charge auto-tuning involves using a CNN to analyze the full CSD to identify all transition lines, then finding the center of the desired regime, say the single-electron regime, all in the image space, therefore no need to iteratively move towards that point. Then directly extract the associated voltage values that would be applied to the gates to reach this regime. This approach fundamentally differs from common patch-based strategies. Patch-based approaches, (as seen in Yon, Durrer, and Czische [91–94]), involve scanning small, localized subsections of the voltage space and classifying each patch for the presence of a transition line and based on this classification, a decision to move in the voltage space is made.

Our approach is possible here at CEA-Leti due to the large amount of charge stability diagrams we have acquired. Leveraging CEA-Leti's ecosystem from measurement infrastructure and expert annotation workflows we have collected and hand-labeled over 1015 high-resolution CSDs across multiple device geometries and operating conditions, far exceeding the publicly available datasets of most other groups. This extensive dataset enables robust training and validation, ensuring our model generalizes across noise profiles and device parameter variations.

Advantages of our Full-Diagram ML-based Approach

Holistic Context: Analyzing the full CSD provides the CNN with a global view of the charge stability landscape. This allows the model to leverage broader spatial relationships and contextual information that might be missed by analyzing isolated patches. For instance, a full-image scan can inherently capture the overall characteristic of the device and learn from features like the slope of the lines, their direction, the spacing between them, or the global curvature of transition lines, which might be ambiguous in small patches. These features are important patterns that ultimately help the model become more robust in the ways discussed below:

- **Robustness to Local Imperfections:** By processing the entire image, the CNN can be more resilient to localized noise, faint lines, or small gaps in transition lines that might otherwise lead to misclassifications in individual patches. The model can "fill in the blanks" or infer the presence of a line based on its global pattern. This could potentially alleviate issues like the interruptions in transition lines caused by the experimental measuring procedure in Czischek *et al.* that led to lower reloading success rates in patch-based methods. We demonstrate in Chapter 6 instance where our model was able to correctly detect a faint line and fill in the blank between broken lines belonging to the same transition.

- **Robustness to Line Misclassifications:** This full image context also helps with differentiating between normal transition lines and what we refer to as spurious lines (see Sec. 4.2.1) which are artifacts due to measurement and they normally have different characteristics but still look like a line which can confuse a model that is solely trained on line detection without any context. For e.g. spurious lines can have a different slope than regular line and therefore by looking at the full-context window, our model was be able to identify the pattern and correctly (un)classify those by merging them into the background.

- **Robustness to Noise:** By leveraging global spatial correlations, the CNN can learn to ignore background fluctuations. Even when the current signal is weak or contaminated by random noise spikes, the network's filters—trained on many examples—can suppress noise and emphasize coherent line structures across the entire voltage range. In practice we see this with very noisy CSDs.

- **Direct Regime Identification:** Instead of an iterative search for a reference point and then counting transitions, a full-image analysis could allow for direct identification of the target charge regime (e.g., the single-electron regime). This could streamline the process by combining detection and localization into a single step, potentially bypassing the "empty-then-reload" sequence if the model is robust enough to identify any arbitrary charge state from a full diagram.

- **Reduced Hard-Coded Exploration Logic:** Patch-based methods often rely on hard-coded exploration algorithms to navigate the voltage space based on local patch classifications. Our full-image approach, removes the need for this exploration by implementing the direct regime identification we discussed in the point before.

- **Physical Features Extraction:** Since our approach consists of detecting all lines in the CSD at once, this means that we have the exact coordinates of each transition line in the diagram and we know how they are positioned relative to each others. Once this information is digitized through our prediction model, we can extract important physical information that helps us better understand the device at hand as well as the physics behind it. For example in our approach we can easily extract the separation between the lines, and the direction of their tilt, as well as their slope etc. These details can be correlated to physical features like capacitance and more.

- **Millisecond Inference for Full-Diagram CNNs** Even though a full-diagram CNN ingests a larger input than a patch-based classifier, its end-to-end inference cost remains in the millisecond range on modern Graphics Processing Units (GPUs), which is still negligible compared to the seconds-to-minutes required for the actual

gate-voltage sweeps and measurements. By contrast, patch-based methods must tile the entire CSD into hundreds or thousands of small windows and run a forward pass on each one—incurring extra data-transfer overhead and per-patch latency—which can easily accumulate to tens or even hundreds of milliseconds just for classification before you even begin post-processing. Since the measurement time dominates (often seconds per diagram), both are effectively "instantaneous" from the experimental workflow perspective, but the global approach still offers a net speedup over patch-based exploration logic alone.

Disadvantages of the Full-Diagram Approach

- **Data Acquisition Time:** Acquiring a full, high-resolution CSD is inherently more time-consuming than acquiring a small patch or a 1D ray. But although this full-image approach with ML inference on top of it might look like a resource intensive procedure, it is important to note that eventually it has to be performed only once per device, as we can completely close the tunnel barriers connecting the QD to the reservoirs after loading n electrons, and we would essentially be working with a closed system for an extended period of time. Moreover, to mitigate the slow acquisition time, we can take a low resolution image or only capture part of the diagram if we know where the area of interest is located. There has also been techniques that were developed to accelerate the data acquisition process and state of the art characterization instruments are also available.
- **Training Data Requirements:** Training a CNN to recognize complex patterns across full CSDs might require a larger and more diverse dataset of labeled full diagrams compared to training a patch classifier. The variability across full CSDs from different devices or experimental conditions could be substantial, necessitating extensive data collection to achieve robust generalization. This could be solved by using transfer learning (see Sec. 2.7.5) where we can take advantage of a pre-trained model and therefore have results with only few images.

3.5 Device Variability and ML-Enabled Feedback

At this point we would like to note that the use of AI to automate the tuning process is required because the current fabricated QD devices exhibit significant run-to-run and wafer-to-wafer variability rather than having a reproducible behavior. Subtle differences in material defects, device geometry and noisy environments mean that a purely signal processing based technique is currently not possible for the auto-tuning of such devices. But in more uniform devices, where variability is reduced, classical signal processing methods can indeed be applied to recover transition lines with reasonable reliability.

However, until fabrication reproducibility reaches that level, ML approaches remain the most robust way to generalize across device-specific quirks and environmental fluctuations. And in order to reach this point of fabrication reproducibility, we first need to have the ability to automatically tune our devices and push them to their limits and this is another reason why ML-based techniques are currently indispensable. Therefore, all of this auto-tuning work is essential for such progress. It is important to mention that all devices will require automated tuning in one way or the other as this is essential to form large arrays of QDs for qubit implementation. However having good quality and reliable QDs would reduce the requirements on the tuning algorithm side and make things easier to deal with. Therefore, in order to achieve scalable QD-based qubits it is necessary for both auto-tuning algorithms and good fabrication quality of devices to improve on the same path. Our full-diagram CNN addresses this dual challenge by:

- **Accelerating Auto-Tuning:** Providing a robust and reliable way into the desired charge configuration as laid down in Sec. 3.4.
- **Driving Fabrication Insights:** by providing feedback and being used as a debugging tool by the experts that are tuning and designing these devices which ultimately improve their quality. This is even more apparent in our specific approach of full-diagram line detection, since we can extract additional physics related information like the distance between the lines, their slope etc. as discussed before.

In this way, our ML framework not only delivers robust and scalable automated tuning but also empowers continuous refinement of QD fabrication, closing the loop between algorithmic calibration and hardware development and pushes QD technology towards scalable, fault-tolerant architectures.

3.6 Conclusion

This chapter has outlined the multi-stage workflow required to configure gate-defined quantum dots, from initial cooldown and coarse topology setting, through charge occupancy tuning, to final qubit calibration. We reviewed classical automation strategies (Baart, Lapointe-Major, Ziegler) that rely on classical image and signal processing techniques or physics-informed heuristics, and we also reviewed ML efforts (Durrer, Czischek, Yon) that employ patch-based inference but remain sensitive to noise and gaps.

Our full-diagram CNN overcomes these limitations by ingesting the entire charge-stability map, combining global spatial context with data-driven robustness. Importantly, this model not only accelerates accurate one-shot charge tuning reducing manual intervention and measurement iterations, but also serves as a feedback mechanism to pinpoint systematic deviations in newly fabricated devices. By correlating prediction errors with

gate geometry or process steps, our framework provides actionable insights to improve the design and fabrication of better devices.

In the next chapters we outline our methodology, starting from data preparation to the model's architecture and implementation. Then we present our results and discuss them.

Chapter 4

Data Acquisition and Preprocessing

In this chapter we describe how the CSDs measurements are converted into the labeled, normalized arrays used for our ML training. We begin with a high-level overview of how the cryogenic measurements are performed and the type of data we collect, then we cover manual labeling of transition lines on exported images, and finally detail the preprocessing pipeline that turns data stored in HDF5 files, into three-channel NumPy arrays plus binary masks ready for training.

4.1 Experimental Setup and Data Acquisition

All data used to train and evaluate our model come from charge stability measurements performed at CEA-Leti. For each device (a gate-defined QD structure), we sweep two gate voltages V_{G1} and V_{G2} over a predefined range, recording the current response $I(V_{G1}, V_{G2})$ using a nearby SET as a charge sensor as we explained in Sec. 2.6. The result is a 2D map of current values, of size $N \times M$ where each pixel value in the CSDcorresponds to the measured current at a particular (V_{G1}, V_{G2}) coordinate.

Measurements are performed in a Cryogenic Wafer Prober (CWP) (Fig. 4.1) at an electron temperature of around $2K$ (base temperature of $0.8K$). We insert into the prober chamber a 300 mm silicon wafer that is patterned with our QD devices. As we perform the 2D sweep on the voltage gates, we record 5 measurements of 100 s each, for every voltage pair (V_{G1}, V_{G2}) and we store the current map in an HDF5 file of shape (N_{scans}, H, W). The result is a stability diagram, recorded as $N_{\text{scans}} = 5$ repeated sweeps. We take 5 measurements in order to use them for current averaging and improve the Signal to Noise Ratio (SNR).

Dataset Composition and Diversity

Our training set consist of 1015 CSDs measured from two distinct mask designs, fabricated over 4 process batches and 7 separate wafers. Within these, we cover both n–type and

Figure 4.1: Wafer Cryogenic Prober used to acquire the CSDs data.

p–type devices (electron and hole trapping respectively) and 9 different gate-pattern geometries. Therefore our dataset consists of measurements with wide variety which makes it ideal for training a robust ML model, and offer a good, unbiased test of that model which would also help us validate its generalizability.

To illustrate the full hierarchy (mask → polarity type → batch → wafer → design), see Table 4.1 below.

Table 4.1: Hierarchical breakdown of CSD counts by mask, polarity, batch, wafer, and design

Mask	**Subgroup**	**Count**
Mask I	n–type, B1 → Wafer i → Design A	83
	n–type, B1 → Wafer i → Design B	61
	n–type, B1 → Wafer i → Design C	81
	n–type, B1 → Wafer i → Design D	147
	n–type, B1 → Wafer i → Design E	138
	n–type, B1 → Wafer i → Design F	104
	n–type, B1 → Wafer ii → Design G	20
	n–type, B1 → Wafer iii → Design F	38
	n–type, B1 → Wafer iii → Design G	23
Mask II	n–type, B2 → Wafer iv → Design H	10
	n–type, B4 → Wafer iv → Design I	196
	p–type, B2 → Wafer vi → Design H	61
	p–type, B3 → Wafer v → Design H	19
	p–type, B3 → Wafer vii → Design H	34

By spanning two mask layouts, both polarities, multiple batches and wafers, and nine device designs, our dataset embodies the process variability encountered in production-scale quantum-dot fabrication. Because each level introduces its own variability in lithography, oxide thickness, and device yield, this broad coverage ensures our model must generalize across real-world fabrication spreads making it robust to such variability.

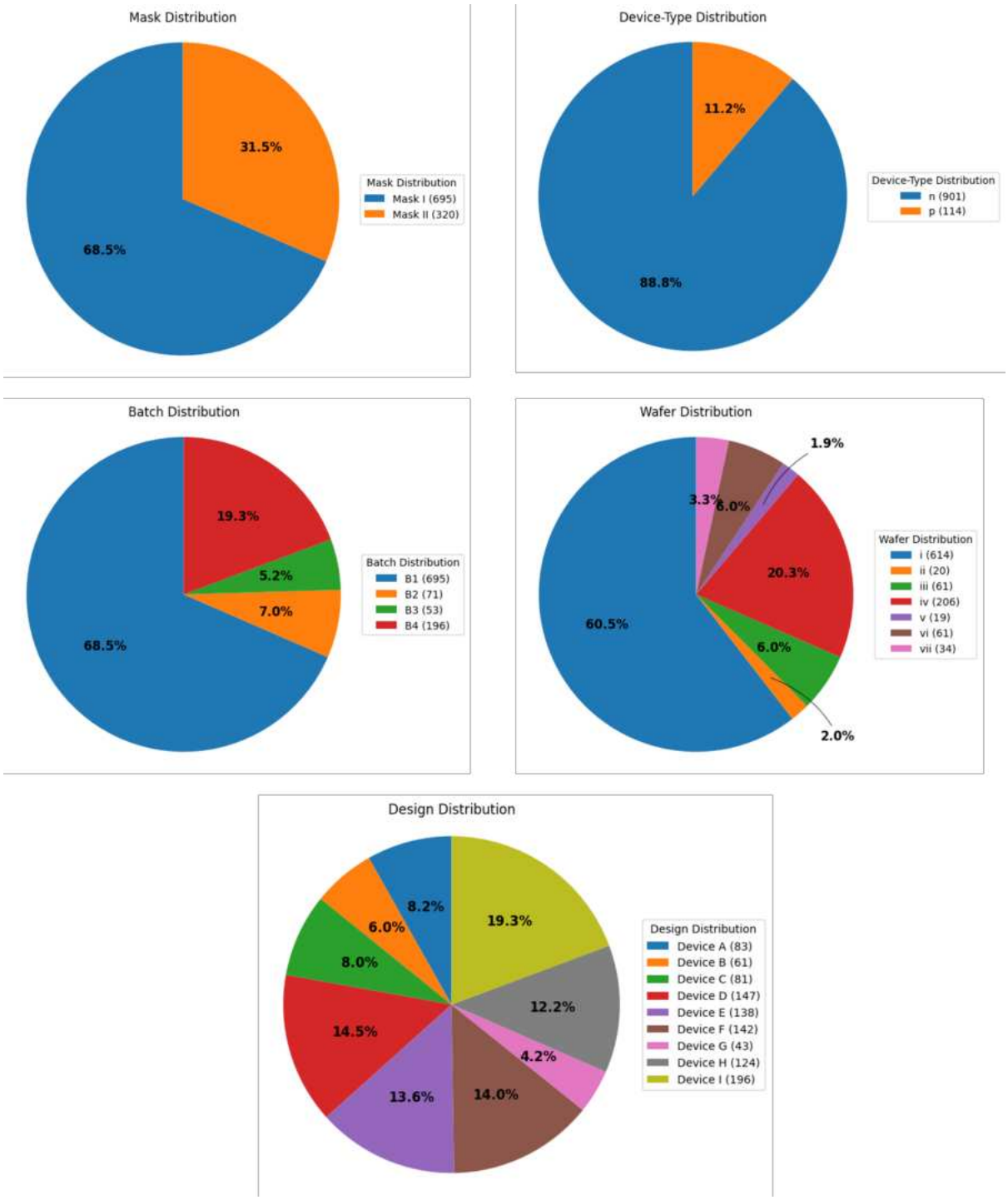


Figure 4.2: Distribution of our CSDs data across mask designs, device polarity, batches, wafers, and device design respectively from top left to bottom right

Summary: *Total CSD measurements:* 1015

- *Mask designs:* Mask I (695), Mask II (320)
- *Device polarity:* n–type (901), p–type (114)
- *Batches:* B1 (695), B2 (71), B3 (53), B4 (196)
- *Wafers:* i (614), ii (20), iii (61), iv (206), v (19), vi (61), vii (34)
- *Geometries (designs):* Design A (83), Design B (61), Design C (81), Design D (147), Design E (138), Design F (142), Design G (43), Design H (124), Design I (196)

4.2 Data Labeling

At this stage we have the raw current measurements that are represented by our charge stability diagrams. Now we need to label this data, which is an essential step in any supervised learning technique. *Data labeling* is the process by which human experts annotate each pixel in an image with the correct class (e.g. "transition line" vs. "background" in our case), thereby providing the target outputs that guide the model's training. Basically, the loss function is calculated by comparing the values of the labeled data (which we refer to as the ground-truth) with the value that the model predicts, and based on that loss, the model will optimize its parameters (see Sec. 2.7.4).

The labeling step is really important because this is where we transfer the domain knowledge of the expert to the ML model by encoding this knowledge into masks which will be used as ground-truth by the model to learn from. Therefore we were extra careful during this stage and performed multiple labeling runs on the same measurements with different experts taking part in this process in order to create high-quality labels that would not bias our model's predictions due to systematic labeling errors.

In our case, the goal is to detect transition lines in CSDs, therefore labeling here consist of manually selecting those lines, and then we can create a binary mask from this data where a pixel value of say 1 represent a line and a pixel value of 0 represent the background or no line. In order to label our current measurement data, first we need to plot them as a charge stability diagram in which the transition lines would be visible to the human eye and ready to be labeled. Therefore labeling takes place on PNG images that we export from the HDF5 files, and is performed by a tool that was custom built in-house by the research team.

To ensure consistency and to cover any potential wiggles in the physical lines, we standardized the thickness of the annotated lines. Each selection is drawn with a fixed width that is wide enough to capture slight deviations but narrow enough to avoid merging nearby features. To further assist in the labeling process, after an initial model was trained on a small labeled set, we integrated this model into our custom labeling tool and ran model inference on new diagrams and presented its pre-segmented lines as suggestions.

Experts then corrected or confirmed these, dramatically accelerating the throughput of high-quality labels. We also employed classical image processing techniques to help facilitate this labeling process by making faint lines more visible to the human eye, among these techniques we implemented gradient-based filters to highlight edge features, and contrast enhancement techniques.

4.2.1 Type of Transition Lines

Before we label our stability diagrams, we define the classes we used for different types of lines and explain briefly the physics behind each type:

Transition Lines: Sharp, high-contrast boundaries where the QD occupancy changes. The dot's electrochemical potential aligns with the source or drain, producing charge transition.

Stochastic Lines: Low-contrast, flickering features. They appear when an electron tunnels repeatedly between dot and reservoir at rates comparable to the scan speed, causing flickering lines.

Spurious Lines: Non-physical artifacts from electrical noise, abrupt charge re-arrangements in nearby gates, or probe-card pickup; these do not correspond to actual dot transitions and must be excluded from training labels.

Examples are shown in Fig. 4.3, with colors marking each type. Red represents spurious lines, green lines for stochastic lines, and blue for normal transition lines.

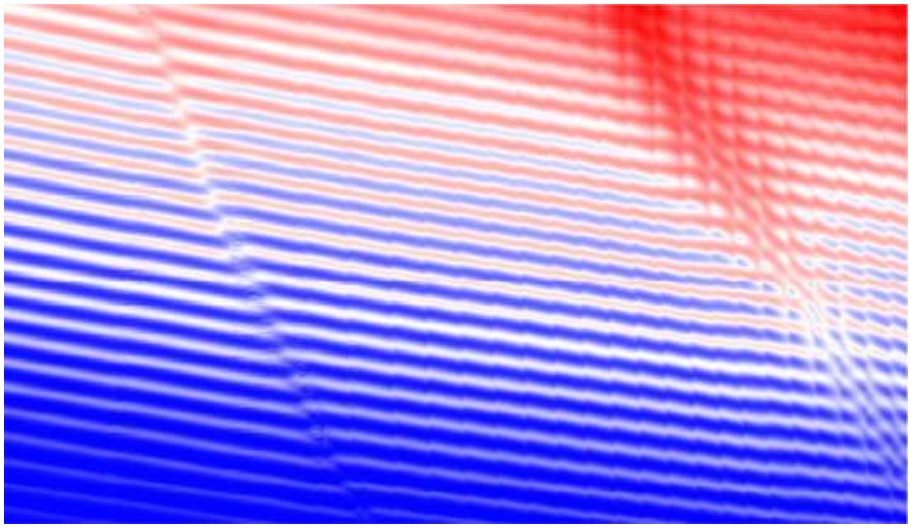
All lines represent a charge transition, only the spurious line does not represent an actual transition and we are not interested in detecting it. Nonetheless, we labeled our lines in a multi-class fashion to be flexible for future implementations like if we want to detect spurious lines so we can remove them from the diagrams.

4.2.2 Labeling Workflow

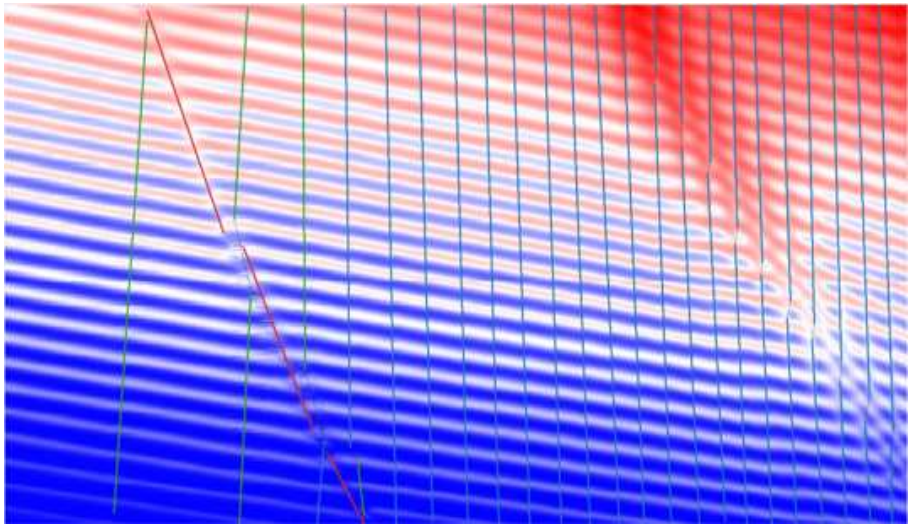
To generate our ground-truth masks, we use a custom GUI tool in which we do the following:

1. Load the PNG of the current map (charge stability diagrams).
2. Choose the appropriate *class* tag for each line segment (see Sec. 4.2.1 for definitions).
3. Draw lines along each segment.
4. Save, per image, a CSV that contains the coordinates of each line's endpoints, with columns:

$$\{\texttt{line\_id}, \texttt{x1}, \texttt{y1}, \texttt{x2}, \texttt{y2}, \texttt{class}\}.$$



(a) Unlabeled CSD



(b) Labeled CSD. Blue lines represent regular charge transitions, green represent stochastic lines, and red for spurious lines.

Figure 4.3: The labeling process of stability diagrams

These labels in the CSV will later be rasterized into binary masks of shape (H, W), where the pixel value at each coordinate follow:

$$Y(i,j) = \begin{cases} 1, & \text{pixel } (i,j) \text{ lies on a transition line segment,} \\ 0, & \text{background and spurious lines} \end{cases} \tag{4.1}$$

These CSVs form the basis for all subsequent mask generation.

4.3 Data Processing

Now that we have performed data acquisition and labeled our data, we are ready to process it to make it ready for training. Our neural network expects a three-channel input of shape $(H, W, 3)$ and corresponding binary masks. (We will explain the reason behind the 3-channel requirement when we discuss the architecture of our model in Chapter 5) The preprocessing pipeline comprises:

- Generating a three-channel array for each diagram which contains the current measurement and will be fed to the model as input.
- Generating binary masks for each diagram which will be our ground-truth, based on which the model will calculate the loss function and optimize its parameters.
- Generating a mask highlighting the location of the single charge regime which will be used during model evaluation to calculate the number of successful detections.

4.3.1 Statistical Feature Extraction

Our raw current data consist of N repeated scans, each giving a 2D-array $I_k(i,j)$ for $k = 1, \ldots, N$ (in our dataset $N = 5$). This is equivalent to having 5 channels per CSD with each measurement sample representing a channel. But as we discussed earlier our model requires only three channels. So the options we have are to either select three of these five sample measurements and use them as our data for training, but this way we are not taking advantage of all the measurements that we have done on the device. Therefore, the best way to take advantage of all the measurements while not stacking redundant channels of the same value is to extract statistical features out of the data and use these features for training.

We select the following features that we calculate on each CSD across the 5 sample measurements that we did: per-pixel mean, median and standard deviation. Therefore, from each raw dataset $I_k(i,j)$ $(k = 1 \ldots 5)$, we compute three per-pixel statistical features:

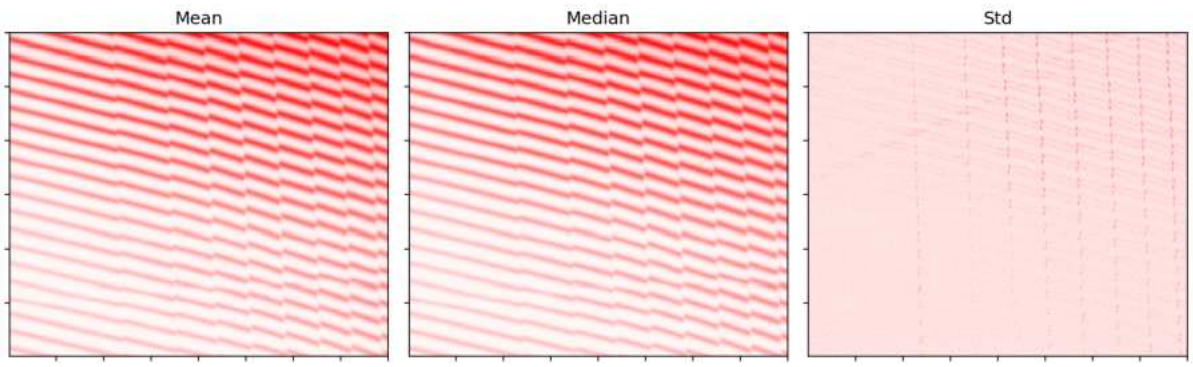


Figure 4.4: Visual representation of the three statistical features (mean, median, and standard deviation) which were calculated from the measurement samples. Each subplot contains a single channel. For better visualization we did not plot them in grayscale. By stacking these features together we get a single three-channel image containing a statistical representation of all measurements.

$$\mu(i,j) = \frac{1}{N}\sum_{k=1}^{N} I_k(i,j), \qquad \text{(mean)}$$

$$\tilde{I}(i,j) = \text{median}\Big\{I_1(i,j), \ldots, I_N(i,j)\Big\}, \qquad \text{(median)}$$

$$\sigma(i,j) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\Big(I_k(i,j) - \mu(i,j)\Big)^2}, \qquad \text{(standard deviation).}$$

We stack these into a 3-channel array:

$$X(i,j) \;=\; \Big[\, \mu(i,j),\; \tilde{I}(i,j),\; \sigma(i,j)\Big] \;\in\; \mathbb{R}^{H\times W\times 3}.$$

Which represent a statistically rich CSD and is our input data to the ML model.

The three channels provide complementary information: the mean enhances strong transitions, the median suppresses outliers, and the standard deviation highlights noise and stochastic features. This multi-channel representation is crucial for our model to distinguish between true transitions and artifacts.

Naming conventions:
- $I_k(i,j)$: raw current scans.
- $X(i,j)$: the three-channel feature image.
- $Y(i,j)$: binary masks.

4.3.2 Normalization

In the previous step, we have calculate statistical features from raw current measurements and stacked them into three-channel. But before we can train on this 2D map of current map, we need to normalize these values. *Normalization* is important to reduce sensitivity to variations, improve numerical stability, and accelerate model convergence. It also

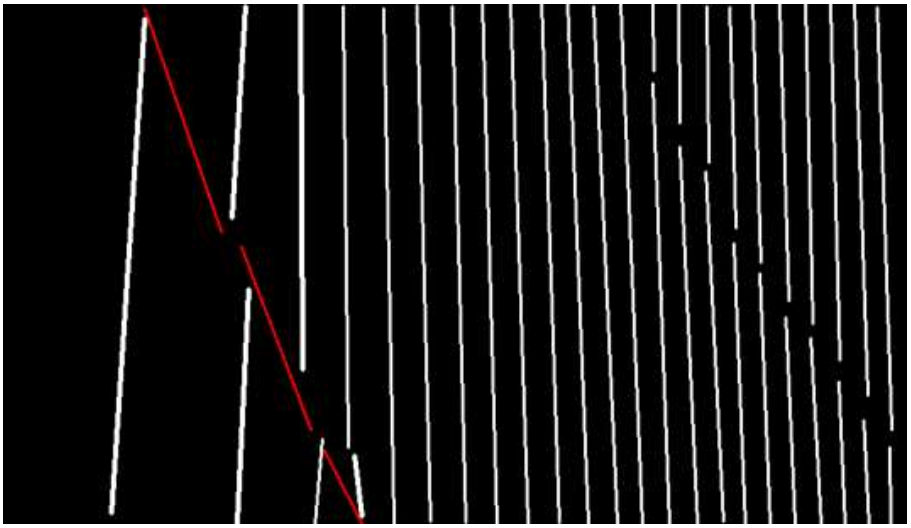


Figure 4.5: Ground-Truth mask generated from the manual labeling done on Fig. 4.3

prevents channels with large absolute values (e.g. high currents) from dominating the training.

Each channel of X is independently min–max normalized to $[0, 1]$ so that all inputs share a common scale:

$$X'_c = \frac{X_c - \min(X_c)}{\max(X_c) - \min(X_c)},$$

4.3.3 Mask Generation

Since our goal is to detect all lines which consist a charge transition, we decide to consider regular and stochastic lines as the foreground (pixel value 1) while spurious lines gets merged into the background (pixel value 0). The only difference we employ between regular and stochastic transition lines is that we draw a slightly thicker polyline in the case of stochastic transitions, to make it easier for the model to map the features of the wide stochastic line from the input image when comparing it to the ground truth.

Using the CSV of labeled lines that we introduced in Sec 4.2, we rasterize each segment onto a binary mask of shape (H, W). For each `line_id` that belongs to a regular or stochastic line, we draw a single continuous polyline from the endpoints delineated by the four coordinate points. We end up with a binary mask $Y(i, j)$ having the same dimensions as the normalized, three-channel stability diagram $X(i, j)$, and together, these will form the input to our ML model.

4.3.4 Single Charge Regime Mask Generation

We also generate a binary mask where we fill in the area of interest with pixel value equal to 1, and the remaining is set as background, so pixel value 0. The area of interest in

our case is either the single electron or hole regime, therefore depending on the device polarity which is known during measurement, we select the first two lines from the left for the case of an n-type device (which is equivalent to the lines having the lowest gate voltage) and the first two lines from the right for p-type (highest gate voltage).

This mask is not used during training, it is only used to check that our detection was successful and was able to locate the region of interest. This mask acts as a ground-truth for the single charge regime. Therefore, we can think of our model as performing two detections, the first is to predict all transition lines in an unseen CSD and the second is to locate the single charge regime. We will talk more about this in the next chapter.

4.4 Conclusion

In this chapter, we have presented the end-to-end workflow for acquiring, labeling, and preprocessing our CSD data into normalized, multi-channel arrays and corresponding ground-truth masks. These prepared datasets form the foundation for our ML approach. In the next chapter (Chapter 5), we will describe the design, training, and evaluation of the neural network that learns to detect and classify transition lines in these diagrams.

Chapter 5

Implementation of the ML-Based Charge Tuning Pipeline

In the previous chapter we detailed our data preparation pipeline, from measuring and labeling the data to performing the necessary pre-processing operations to make our data compatible with the model we are building. At this stage we have data that consists of CSDs and ground-truth masks ready to be used for supervised learning (see Sec. 2.7.1). In this chapter we describe the full training and inference pipeline. We begin by framing the line-detection task as a pixel-wise segmentation problem, then present our network architecture (a U-Net with ImageNet-pretrained encoder), detail the training protocol (data augmentation, GroupKFold cross-validation, loss functions, optimizer), and finally explain our post-processing steps to extract gate-voltage recommendations for the single electron (or hole) regime . A high-level overview is shown in Fig. 5.1.

5.1 Problem Formulation

We treat charge transition line detection as a *pixel-wise binary segmentation* task. Given an input CSD

$$X \in \mathbb{R}^{H \times W \times 3},$$

where the three channels are the pixel-wise mean, median, and standard-deviation maps across multiple voltage sweeps (Sec. 4.3.1), the neural network f_θ predicts

$$\hat{Y} = f_\theta(X) \ \in \ [0,1]^{H \times W},$$

such that $\hat{Y}_{ij}$ estimates the probability that pixel (i, j) belongs to a transition line. During training, we compare $\hat{Y}$ against the ground-truth binary mask $Y \in \{0,1\}^{H \times W}$ (see Eq. 4.1) via the Dice loss (see Eq. 5.1) which we aim to minimize and update the weights of the model accordingly.

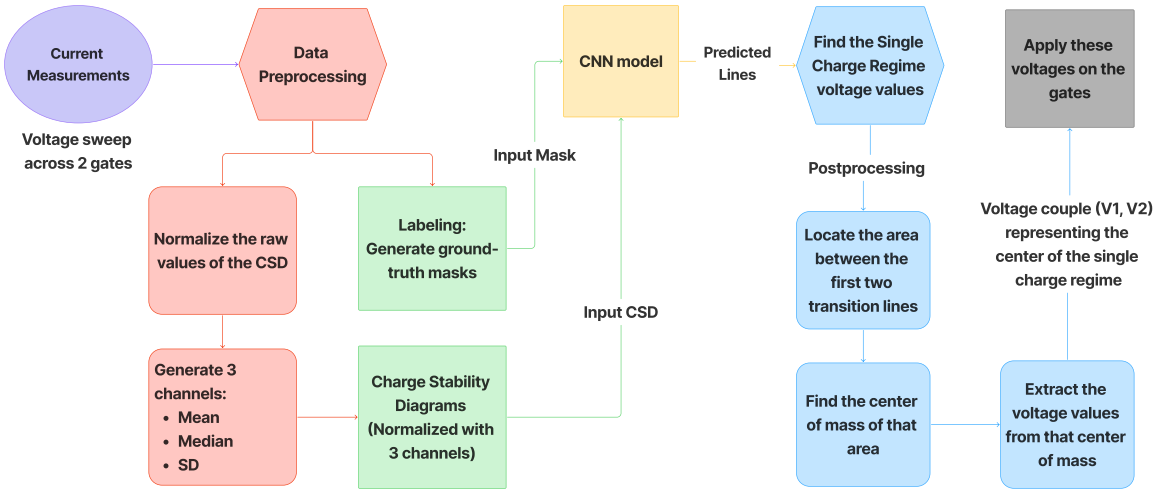


Figure 5.1: **End-to-end offline-tuning pipeline.** We start with current data measurement (in purple) by sweeping the voltage on the gates as explained in Sec. 2.6. Then we process the data (red) by normalizing our raw measurements and calculating the three statistical channels from those measurements (see Sec. 4.3. After labeling, we end up with input mask and CSD (green) to be fed to our ML model (yellow). The outcome of our model is a prediction of the transition lines in the stability diagram. From this prediction mask we perform the necessary postprocessing steps explained in 5.4 and we find the single electron or hole regime (blue). Finally we apply the gate voltages of that regime as we extract them from the prediction mask (gray).

5.2 Model Architecture

In this section, we present the overall architecture of our U-Net–style segmentation model [97], built upon a lightweight MobileNetV2 encoder [98] pretrained on ImageNet's large database [99]. This combination provides both efficient feature extraction and precise spatial localization. MobileNet variants already have weights trained on millions of natural images like the ones trained on ImageNet. Although natural images differ from CSDs, the early convolutional layers learn geometric features and texture that are relevant to most image-based tasks. Initializing with these weights accelerates convergence and improves generalization.

We first describe the encoder backbone, then detail the decoder design including skip connections and upsampling, and finally specify the output head and loss function. Here is an overview of our structure:

- **Encoder:** MobileNetV2 backbone pretrained on ImageNet, providing rich, hierarchical feature maps while preserving efficiency.
- **Decoder:** Symmetric upsampling path with skip-connections at each resolution, using Conv–BatchNormalization–ReLU blocks to recover spatial detail.
- **Output Head:** A 1×1 convolution followed by sigmoid produces the per-pixel probability map $\hat{Y}$.

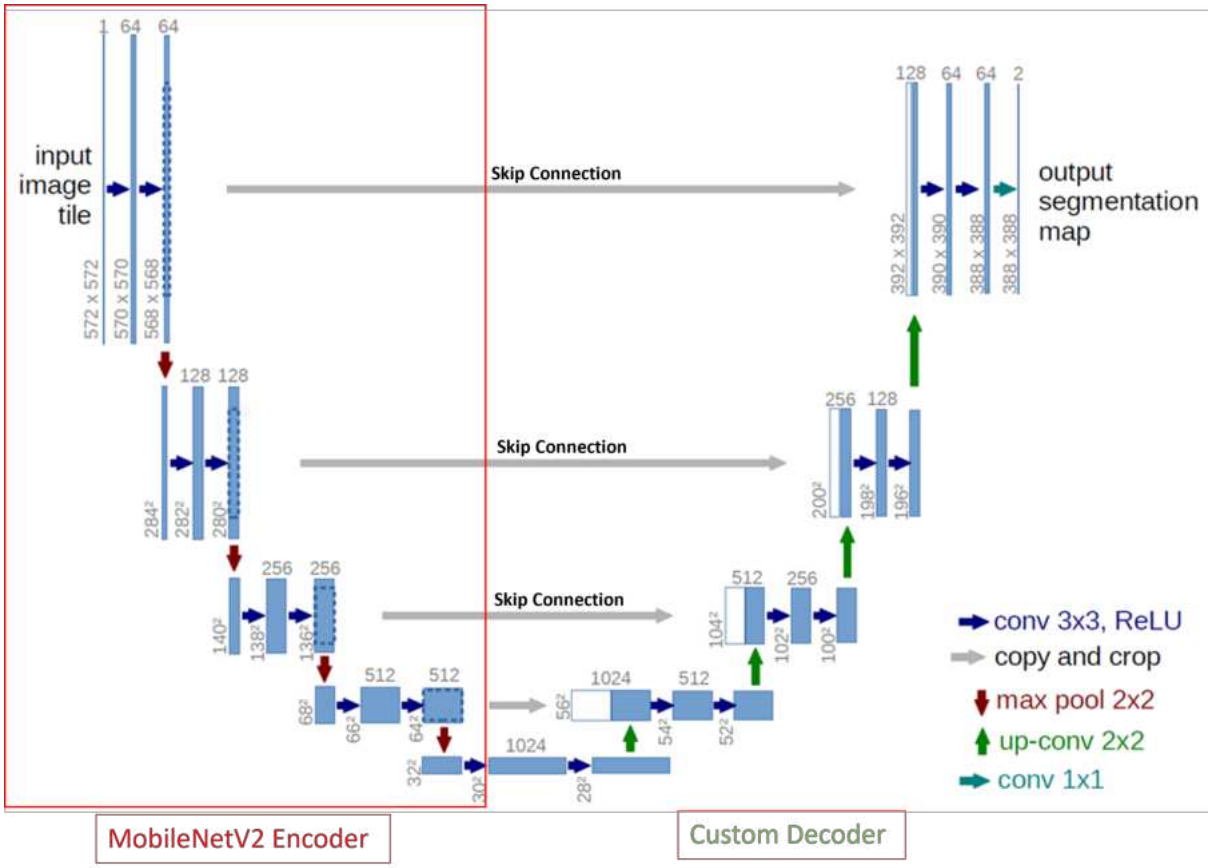


Figure 5.2: Original U-net architecture with slight modification. We kept the structure and numbers of the original paper [97] we only highlighted how our implementation adapts the U-net architecture by using a pre-trained MobileNetV2 as the encoder and a custom decoder which we describe in Sec. 5.2.2. The encoder and decoder are highlited in red and green respectively. Blue boxes corresponds to feature maps having multiple channels which are denoted on top of each box. The spatial dimensions of the image are presented on the lower left edge of each box. In our case the starting input has a dimension of 1024x1024. The gray arrows are skip connections as highlighted in the legend, and the white boxes are the copied feature maps which gets concatenated during upsampling. (Adapted from [97])

5.2.1 Encoder: MobileNetV2 Backbone

The encoder is based on the *MobileNetV2* architecture developed by Google. We adopt the pre-trained ImageNet weights and remove the classification head, retaining all convolutional feature extractors. Given an input tensor of size $H \times W \times 3$, the encoder produces a hierarchy of feature maps at multiple spatial resolutions. We extract the following intermediate activations to serve as skip-connection sources:

- `block_1_expand_relu` $(\frac{H}{2} \times \frac{W}{2} \times 144)$
- `block_3_expand_relu` $(\frac{H}{4} \times \frac{W}{4} \times 192)$
- `block_6_expand_relu` $(\frac{H}{8} \times \frac{W}{8} \times 288)$

- `block_13_expand_relu` $\quad (\frac{H}{16} \times \frac{W}{16} \times 816)$

We denote the deepest encoder activation (`block_13_expand_relu`) by $\mathbf{F}_{\text{bottleneck}}$.

5.2.2 Decoder: Upsampling, Skip Connections, and Feature Fusion

The decoder follows the standard U-Net paradigm. Starting from $\mathbf{F}_{\text{bottleneck}}$, we perform a cascade of four upsampling stages. At the i-th stage (from deepest to shallowest):

$$\mathbf{U}_i \;=\; \text{UpSample}_2(\mathbf{U}_{i-1}), \quad \mathbf{S}_i \;=\; \text{Skip}_i, \quad \mathbf{C}_i \;=\; \text{ConvBlock}\Big([\mathbf{U}_i,\, \mathbf{S}_i]\Big),$$

where

- UpSample_2 denotes a 2×2 nearest-neighbor upsampling which doubles the spatial resolution.
- Skip_i is the encoder activation at resolution level i. It does a concatenation with the corresponding encoder feature map to restore fine details lost during downsampling.
- $[\cdot, \cdot]$ indicates channel-wise concatenation,
- ConvBlock consists of two repetitions of

$$\text{ReflectionPadding2D}(p = 1) \;\to\; \text{Conv2D}(3 \times 3,\, f_i) \;\to\; \text{BatchNorm} \;\to\; \text{ReLU}.$$

Reflection padding avoids border artifacts without introducing zero-padding distortions.

Here, $f_i \in \{64, 48, 32, 16\}$ are the decoder channel widths at each stage, chosen to gradually refine spatial details while controlling model complexity.

5.2.3 Final Prediction Layer

After the last upsampling stage, we apply a 1×1 convolution to reduce to a single-channel feature map

$$\mathbf{Z} \;=\; \text{Conv2D}_{1\times1}(\mathbf{C}_4) \quad \in\; \mathbb{R}^{H \times W \times 1},$$

followed by a sigmoid activation:

$$\hat{\mathbf{Y}} \;=\; \sigma(\mathbf{Z}),$$

which yields per-pixel probabilities for the binary segmentation mask.

5.2.4 Loss Function: Dice Loss

We optimise the network using the *Dice loss*, which directly maximises the overlap between prediction and ground truth. Given prediction $\hat{Y}$ and binary ground truth Y, the Dice coefficient is

$$\text{Dice}(Y, \hat{Y}) \;=\; \frac{2\,\sum_i Y_i \hat{Y}_i + \epsilon}{\sum_i Y_i + \sum_i \hat{Y}_i + \epsilon},$$

as introduced in Sec. 2.7.6, and the corresponding loss is

$$\mathcal{L}_{\text{Dice}} \;=\; 1 \;-\; \text{Dice}(Y, \hat{Y}) \tag{5.1}$$

Here, ϵ is a small constant for numerical stability. This loss is well-suited for highly imbalanced segmentation tasks like in our case, and encourages maximization of overlap rather than per-pixel accuracy. We compile the model with the Adam optimizer (learning rate 10^{-4}) and the Dice loss.

The resulting architecture combines the computational efficiency of MobileNetV2 with the strong localization capabilities of U-Net decoders, producing a lightweight yet accurate segmentation model.

5.3 Training Procedure

In this section we detail the procedures used to train our segmentation model, including our computational environment, how we split the data to avoid leakage, input preprocessing (resize & padding), and on-the-fly data augmentation.

5.3.1 Computational Environment

All model training and large-scale inference were performed on the TGCC (Très Grand Centre de Calcul du CEA) "Irène Joliot Curie" high-performance computing (HPC) cluster. This facility provides both CPU-only nodes and GPU-accelerated nodes; for our work we exclusively used the NVIDIA Tesla V100 GPUs to leverage parallelized deep learning workloads.

5.3.2 Data Splitting & Cross-Validation

To prevent *data leakage* from images originating on the same physical device or cooldown, we employ `GroupKFold` cross-validation. In GroupKFold, samples are partitioned into K folds such that all samples sharing the same `group_id` appear in exactly one fold, either training or validation. We construct the `group_id` to be a unique identifier for images that were measured from the same devices (same wafer, die, and gates used). We set

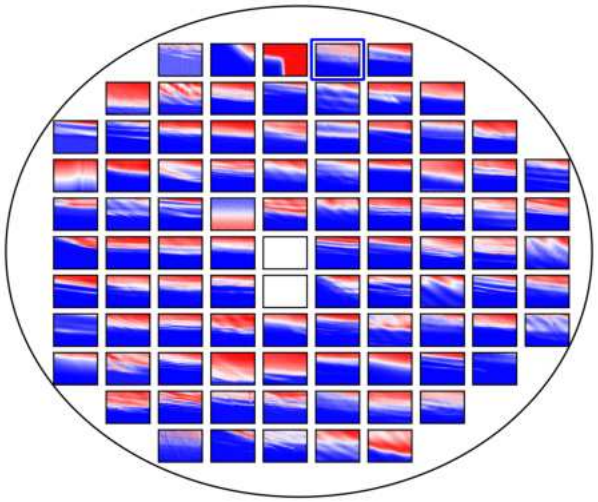


Figure 5.3: Representation of how every CSD comes from a die on a wafer

$K = 3$ and group by device, yielding three pairs of splits each with two-thirds of devices for training and one-third for validation.

Inference on Similar Images. At test time, it is sometimes common to encounter images from the same device, batch, or wafer. Although these share the same physical features, each measurement is taken at a different timestamp under different conditions and after cooling and system re-initialization, which introduces fresh noise realizations, spurious lines, and slight drift in transition line positions. Consequently, performing inference on multiple images from the same device does not constitute a bias, since the model has never seen the exact measurement conditions during training. We refer to Fig. 5.4 for a concrete example of different stability diagrams measured from the same physical device but under different conditions and timing, showing clear variation in noise, contrast, and illumination.

5.3.3 Input Preprocessing: Resize and Padding

Our raw charge stability diagrams come in a variety of resolutions and aspect ratios (aspect ratio $= \frac{\text{width}}{\text{height}}$), which complicates batching and GPU memory allocation. We compared three preprocessing strategies:

1. **Pure Resize:** We force every image to a fixed canvas ($H_{\text{target}} \times W_{\text{target}}$) by resizing both dimensions independently (e.g. with bilinear or nearest-neighbour interpolation).

   ✓ Very fast and simple to implement.
   × Unless each raw image already shares the same aspect ratio, this "stretch-to-fit" approach distorts the geometry: lines that were straight can become curved

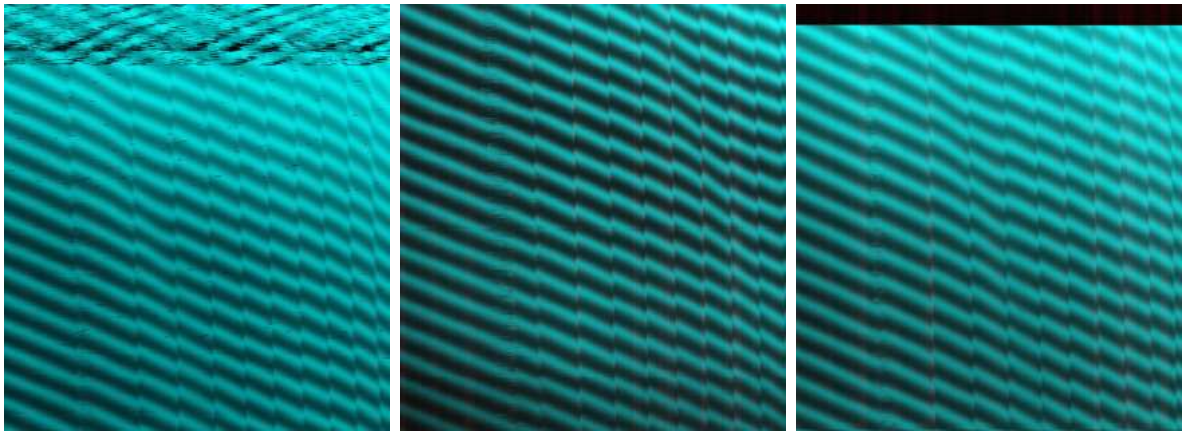


Figure 5.4: Three charge stability diagrams acquired from the same physical device under different cooldowns and measurement conditions. Despite identical device geometry, variations in thermal noise, contrast, and drift in transition lines are evident between measurements, demonstrating that each diagram represents a distinct sampling of the device's state rather than redundant data.

or shifted. Fine features may be blurred or aliased depending on whether we downsample (loss of high-frequency detail) or upsample (interpolation blur).

2. **Pure Padding:** We leave each image at its native size (H, W), then embed it in a larger $(H_{target}, W_{\text{target}})$ canvas by adding constant-value borders (black, mirror, or reflection).

   ✓ Preserves every original pixel and exact aspect ratio; no interpolation means no blur or aliasing.
   × Can create large blank regions that dominate convolutional receptive fields therefore wasting computation on padding.

3. **Resize & Pad (Chosen):** We first compute a scale factor

$$s \;=\; \min\Big(H_{\text{target}}/H,\; W_{\text{target}}/W\Big),$$

   resize the image to $\big\lfloor sH \big\rfloor \times \big\lfloor sW \big\rfloor$ (using nearest-neighbor to preserve edge sharpness), then pad the short side with black pixels to exactly $(H_{\text{target}}, W_{\text{target}})$.

   ✓ Maintains original aspect ratio and avoids extreme blank regions.
   ✓ Minimizes interpolation artifacts by only scaling as much as necessary to fill the longer side.
   ✓ Simplifies downstream coordinate recovery: by storing s and the top/left padding offsets, one can map any detected pixel (x', y') back to its original image coordinate

$$\big(x, y\big) = \Big(\tfrac{x'-x_{\text{pad}}}{s},\; \tfrac{y'-y_{\text{pad}}}{s}\Big), \tag{5.2}$$

ensuring accurate extraction of physical voltage values from single charge transitions.

Impact on Single Charge Voltage Extraction: Since our ultimate goal is to read off gate voltages corresponding to pixels inside the single charge regime, any geometric transform that shifts or blurs pixel locations can bias the measurement. Our resize & pad routine uses a uniform scale s and known padding amounts, and we invert this transform to recover the original pixel grid exactly, thereby guaranteeing that voltage coordinates remain faithful to the raw data.

Alternative: Fully Convolutional Networks. A fully convolutional approach would accept arbitrary input sizes, but complicates batching (batch size=1 or grouping by size) and GPU memory planning. We reserve this avenue for future work.

5.3.4 Data Augmentation

To enhance robustness to device-specific noise and slight geometric variation, we apply the following on-the-fly augmentations during training only:

- **Horizontal Flips**: captures possible mirroring of stability diagrams resulting from sweep direction reversals or different device polarity.
- **Small-Angle Rotations**: accounts for slight tilts in transition lines.
- **Random Scaling/Cropping**: simulates variations in gate-voltage ranges or zoom-levels, preventing scale-sensitive bias.
- **Color Space Transformations**:
  - *Brightness Adjustment*: simulate different illumination levels by varying pixel intensities.
  - *Contrast Adjustment*: enhance or reduce contrast to help the model recognize lines under varying clarity.
  - *Saturation Adjustment*: makes the model robust to diverse color intensities by altering saturation levels.

Each batch is generated with random parameter draws, ensuring the model encounters diverse patterns with geometric and color transformations, improving generalization to unseen devices and cooldown conditions.

5.4 Prediction and Postprocessing

After having trained the model on our stability diagrams, we are ready to perform predictions on new unseen data with the goal of finding the single electron or hole regime.

We detail below the sequential steps that we perform starting from model inference to get our prediction mask, until the last step of the postprocessing which is locating the single charge regime and extracting the gate voltage values that we need to apply to reach that regime.

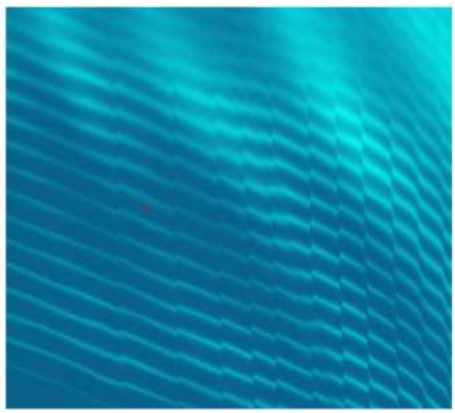
We give a broad overview of the steps then we dive into each of them:

1. **Forward Pass:** Input the normalized, padded image and compute per pixel probabilities $\hat{Y}$.
2. **Thresholding:** Binarize $\hat{Y}$ at a threshold of $\tau = 0.75$.
3. **Undo Padding & Resize:** Reverse the padding and resize that we applied before inference, to map the mask back to the original voltage sweep dimensions.
4. **Morphological Cleanup:** Perform closing operation to connect broken line segments.
5. **Filter Spurious Detections:** Perform area-based dynamic thresholding to remove small sized spurious detections.
6. **Single Charge Regime Identification:** Extract the two extreme transition lines and find the center of the single electron (or hole) regime:
    - Locate the two lowest-voltage transition lines (or highest for holes trapping) in the clean mask.
    - Define the polygonal region between them as the candidate single electron (or hole) regime.
    - Compute the center of mass of this polygon and extract the gate-voltage values (V_{G1}^*, V_{G2}^*) which corresponds to the center of the single charge regime.
7. **Success Flag**: In case we are doing offline auto-tuning and not inferring on unlabeled image, therefore, we can confirm if the predicted single charge regime's center matches with our labeled ground truth mask, which validates our detection as successful or not.

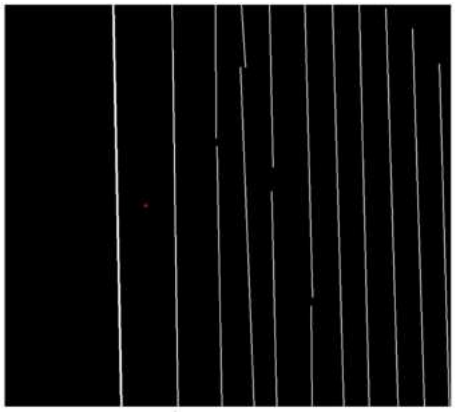
We visualize these steps in Fig. 5.5 in which starting from the input CSD we perform the forward pass, thresholding, then the morphological operations as well as filtering, and finally detecting the single charge regime and comparing it to the ground truth mask associated to that diagram.

Forward Pass

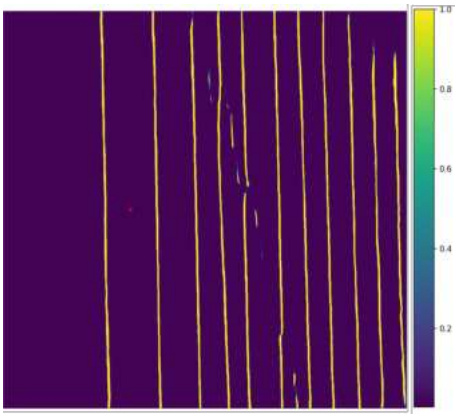
We input the normalized, resized and padded image into our trained model, and we get as an output the prediction $\hat{Y}$ which is a mask with continuous pixel values between 0 and 1 representing the probability of each pixel being classified as a transition line.



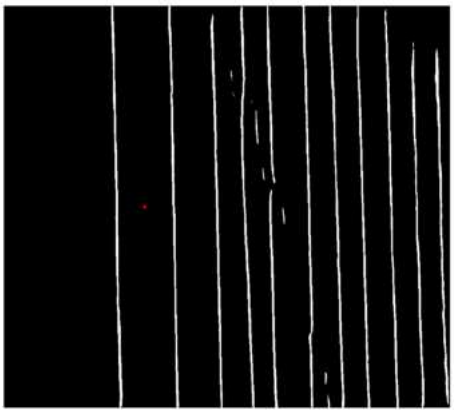
(a) Input Image



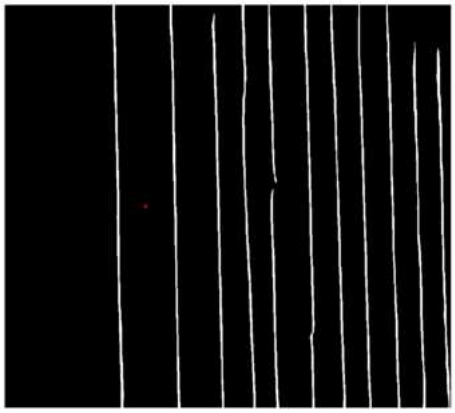
(b) Ground Truth Mask



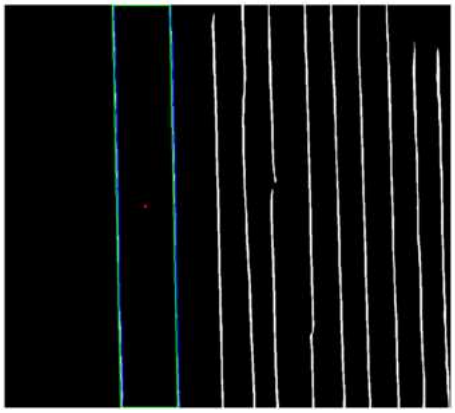
(c) Prediction Probability Map



(d) Binarized Map



(e) Morphology + Filtering



(f) Single Charge Regime

Figure 5.5: **Plot showing inference and the main postprocessing steps.** We chose a diagram in which our model's prediction is not very "clean" in order to highlight the importance of the postprocessing steps we implement, which were discussed in Sec. 5.4. (a) The input CSD in the form of an image. (b) The ground truth mask labeled as discussed in Sec. 4.2 and used for training the model. (c) The raw model's prediction where each pixel is assigned a probability of how confident the model is that this pixel belongs to a transition line. (d) After thresholding the previous probability map based on a fixed value of 0.75 we get a binary map with the white pixels as lines and the black as background. (e) We apply morphological closing and area dynamic filtering to connect broken lines and remove small spurious detections. (f) On the final cleaned binary mask we find the first two transition lines and locate the center of mass of the single electron regime which we represent here by a red square patch overlaid on top of all the images at the same position.

Therefore, given a normalized, resized, and padded CSD $X \in \mathbb{R}^{H\times W\times 3}$, the network outputs

$$\hat{Y} = f_\theta(X) \in [0,1]^{H\times W},$$

Thresholding

Now we need to transform the continuous probability mask we got in the previous step into a binary classification mask with pixel values of 1 if it is part of a transition line and pixel values of 0 representing the background. We achieve this by thresholding the prediction $\hat{Y}$ with a threshold value of $\tau = 0.75$ which would select all pixels with probability value above 75% as transition line pixels, and the rest would become a background.

Mathematically, we convert $\hat{Y}$ into a binary mask $\hat{Y}^\tau \in \{0,1\}^{H\times W}$ via

$$\hat{Y}^\tau_{ij} = \begin{cases} 1, & \hat{Y}_{ij} \geq \tau, \\ 0, & \hat{Y}_{ij} < \tau, \end{cases} \qquad \tau = 0.75.$$

This selects high-confidence pixels as candidate line segments.

Undo Padding & Resize

During our training procedure we performed a resize and padding operation in order to preserve the image's aspect ratio (see Sec. 5.3.3). We also did this operation before the forward pass because our trained model expects images that have the same dimensions as the one we used during training. Now after we have performed inference and got our prediction mask on the resized and padded image, we undo these operations in order to revert back to the original size of the image which is necessary since our goal is to locate the pixel coordinates that corresponds to the voltages required to have a QD in the single charge regime. To do this we store information about the resize factor and the padding values that we used when we performed these operations before feeding our image for inference, and here we reverse them for each image's specific values as per Eq. 5.2.

Morphological Operations

Now we apply some morphological operations on our predicted mask with the goal of:

- Connecting broken lines through morphological closing.
- Filtering out small spurious mis-detection using a dynamic threshold.

We do this postprocessing in order to make the subsequent steps more robust to small pixel-wise errors so they do not carry over.

First, a morphological closing consist of performing a dilation followed by an erosion [100].

$$\hat{Y}^{\tau}_{\text{closed}} \;=\; \left(\hat{Y}^{\tau}_{\text{orig}} \;\oplus\; K\right) \;\ominus\; K,$$

where

$$K \;=\; \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix}}_{20\times 2}$$

is the rectangular structuring element of size 20×2, $\oplus$ denotes morphological dilation, and $\ominus$ denotes morphological erosion.

We use a vertical rectangular kernel of size (20px, 2px) with the goal of bridging small gaps between transition lines that are supposed to be connected. We chose this kernel size because the lines in our diagram are mostly vertical with slight angle deviations, so we are interested in connecting them along the y-axis.

Next, we remove spurious connected components whose area is too small. First, we calculate the area of all connected components in the prediction mask and find the average area per component. This number will vary for every stability diagram, thus why we termed the method dynamic, because it is diagram specific. We then use this value as a threshold to filter out any connected component that is under 75% of that average area.

Let $\{C_k\}$ be the connected components in $\hat{Y}^{\tau}_{\text{closed}}$, with areas A_k. We compute the mean area $\bar{A} = \frac{1}{K}\sum_k A_k$ and set a dynamic threshold to be $T = 0.75\,\bar{A}$. The filtered mask is

$$\hat{Y}^{\tau}_{\text{filt}}(x) = \begin{cases} 1, & \text{if } x \in C_k \text{ and } A_k \geq T, \\ 0, & \text{otherwise.} \end{cases}$$

Single Charge Regime Detection

Now that we have done all the necessary processing on our prediction mask, we are ready to find the single charge regime which is the main goal. From the filtered mask, we sort each connected component based on its x coordinates, and depending on if the device is n-type, meaning that the QD will trap electrons or p-type if it is trapping holes, we select the first two lines our model detected, that have the lowest gate voltage values in the case of an n-type device, or the highest if it is a p-type. These two lines delineate the edges of our single electron or hole regime. Then we form a quadrilateral polygon between the line's extreme points, and we find the center of mass of that polygon which corresponds to finding the center of the single charge regime.

Voltage Extraction

Since we are working with raw current data, when we locate the center of the single charge regime, we just need to extract the coordinate location of that center and these values (V_{G1}^*, V_{G2}^*) would represent the voltage values that should be applied on the gates in order to reach the center of the single electron or hole regime. Therefore in an online tuning approach, these recommended voltages are then directly applied in the experimental control software to set the device into the single-electron regime in one shot.

Success Flag

Now that we have located our single charge regime and extracted an associated voltage values for it, we need to make sure it is the correct one. Since this is an offline test we are working with labeled data, therefore it suffice to test if the voltage value we extracted which is the center of mass of the region of interest, actually falls into the single electron (or hole) regime. During preprocessing, we have created a mask for that task (see Sec. 4.3.4). The mask we have created was generated from the ground-truth labels, and consists of the single charge area filled with pixels of values 1, and the rest is a background with pixel values of 0. Therefore, we can simply check if our detection we did in Sec. 2.6 for the center of mass of that regime, agrees with the ground truth mask we have generated.

If the pixel (i^*, j^*) falls within the true single charge regime (which is represented in our ground-truth segmentation mask), then the detection for that diagram is successful, otherwise it is false. We define this metric as our *Success Flag* which is defined as:

$$\text{Success Flag} \;=\; \frac{\text{Number of images correctly detected}}{\text{Total number of images}} \times 100\% \tag{5.3}$$

Therefore, throughout this thesis we refer to Eq. 5.3 as our offline tuning accuracy where a success means that we accurately detected the center of mass of the single charge regime.

5.5 Evaluation Metrics

At this point it is clear that there are two different evaluation metrics for our task. The first are the regular ML model metrics that represents how well our model is at detecting what we have already labeled (see Sec. 2.7.6), which in our case are transition lines. And the second type is the metric we have defined in the previous section (see Eq. 5.3) which represent the success rate of accurately finding the single charge regime by checking if our detection falls in the region as defined in our ground truth mask. Both metrics relies

on comparing prediction v.s. the labeled data, either pixel-wise line detection or area localization.

Therefore, for each CSD, we record:

- **Success Flag:** True if the predicted single charge regime center falls in the single charge regime region of the ground truth mask.
- **Pixel-Level Metrics:** Dice coefficient, IoU, Pixel Accuracy, Precision, and Recall.

5.6 Conclusion

In this chapter we presented a detailed, modular implementation of our ML-based charge-tuning pipeline. From task formulation and network design through rigorous training on grouped cross-validation folds, to a robust inference and post-processing routine yielding direct gate-voltage recommendations. In the next chapter we present quantitative results of our study and follow-up with a discussion and failure analysis.

Chapter 6

Experimental Results of Offline Auto Tuning

In this chapter we present the experimental evaluation of the offline auto-tuning pipeline developed in this work. Our aim is to quantify the ability of the proposed convolutional segmentation approach to detect transition lines and reliably localize the single-charge regime across a diverse set of devices and measurement conditions. We report both quantitative results (per-fold and per-device design success, and pooled performance) and qualitative examples that illustrate typical successes and failure modes. Finally, we discuss a structured failure analysis and propose concrete mitigation strategies that can be applied in both offline and online tuning scenarios.

6.1 Results and Discussion

This section reports the performance of the proposed single-charge detection pipeline. We start with qualitative evaluation, then we report the quantitative results (per-device design and per-fold). All experiments use 5-fold group cross-validation (203 test images per fold, 1015 images total); the evaluation metric is the "success" flag described in Sec. 5.4 (a detection is counted as successful if the predicted single-charge regime center lies within the tolerance region of the ground truth).

6.1.1 Qualitative Results

The primary objective of this work is to reliably locate the single charge regime in a stability diagram, rather than achieve perfect pixel-wise segmentation of transition lines. Therefore, qualitative evaluation of our model such as looking at the quality of the detected lines as well as the success ratio we described in Eq. 5.3, are very important to help us understand where did the model underperform and give us the feedback that we need in order to make changes or improvement to our pipeline.

Figure 6.1 shows typical model prediction side by side with the original stability diagram and its associated hand-labeled ground truth mask. We show the detection of transition lines, as well as the single electron regime which is highlighted by a green contour and a square red patch locating its center. We overlay this patch on top of both the CSD and the ground truth mask to show accurate localization of the single electron regime. We highlight the accuracy of our model in both detecting the transition lines under difficult conditions as well as successfully locating the single charge regime. In this figure we chose a variety of diagrams from different devices to showcase the variability of our data as well as the robustness of our model to this variability. For example row 1 and 5 show successful detection of all transition lines under considerable horizontal noise, and row number 2 shows a stability diagram with very weak Coulomb peaks and faint transition lines, whereas the stability diagram of the 3rd row has a very week SNR and in some instances vanishing Coulomb peaks but our model had no difficulty detecting most of the lines and was successful in locating the single electron regime. Moreover, our model was for the most part accurate in ignoring spurious lines like the ones you can see in the CSD of row 4. Regarding the final prediction, row 6 contains a mixture of most of these cases in which we have a spurious line on the far left which was correctly ignored by our model, but in addition to that the model successfully detected the first two transitions which were of a stochastic nature, and was not bothered by the horizontal noise and accurately detected all the remaining transition lines. We have many instances of such successful detections across both good and bad quality stability diagrams, see Figures B.1 and B.2 in Appendix B for more examples.

These visualizations are useful in three ways: they (1) validate that the system succeeds on practically relevant examples even if pixel-wise metrics are imperfect, (2) help identify systematic failure modes that are not obvious from aggregate numbers, and (3) guide further improvements in preprocessing, labeling, or postprocessing. Along with this visual representation of the detection, we save the predicted gate voltages that would correspond to that regime in a separate file for every diagram.

6.1.2 Quantitative Results

Aggregate Per-Design Success

Table 6.1 reports the success rate of accurately detecting the single charge regime for each unique device design, aggregated over all five folds. For each design we give the raw counts (successful detections / total diagrams) and the pooled success. Each design consist of a unique gate architecture, and are anonymized as *Design A–Design I*. The inference success rate we present here, is computed as the ratio of CSDs where we correctly located the single charge regime to the total number of diagrams we performed our test on across all folds (see Sec. 5.4 for the success flag metric).

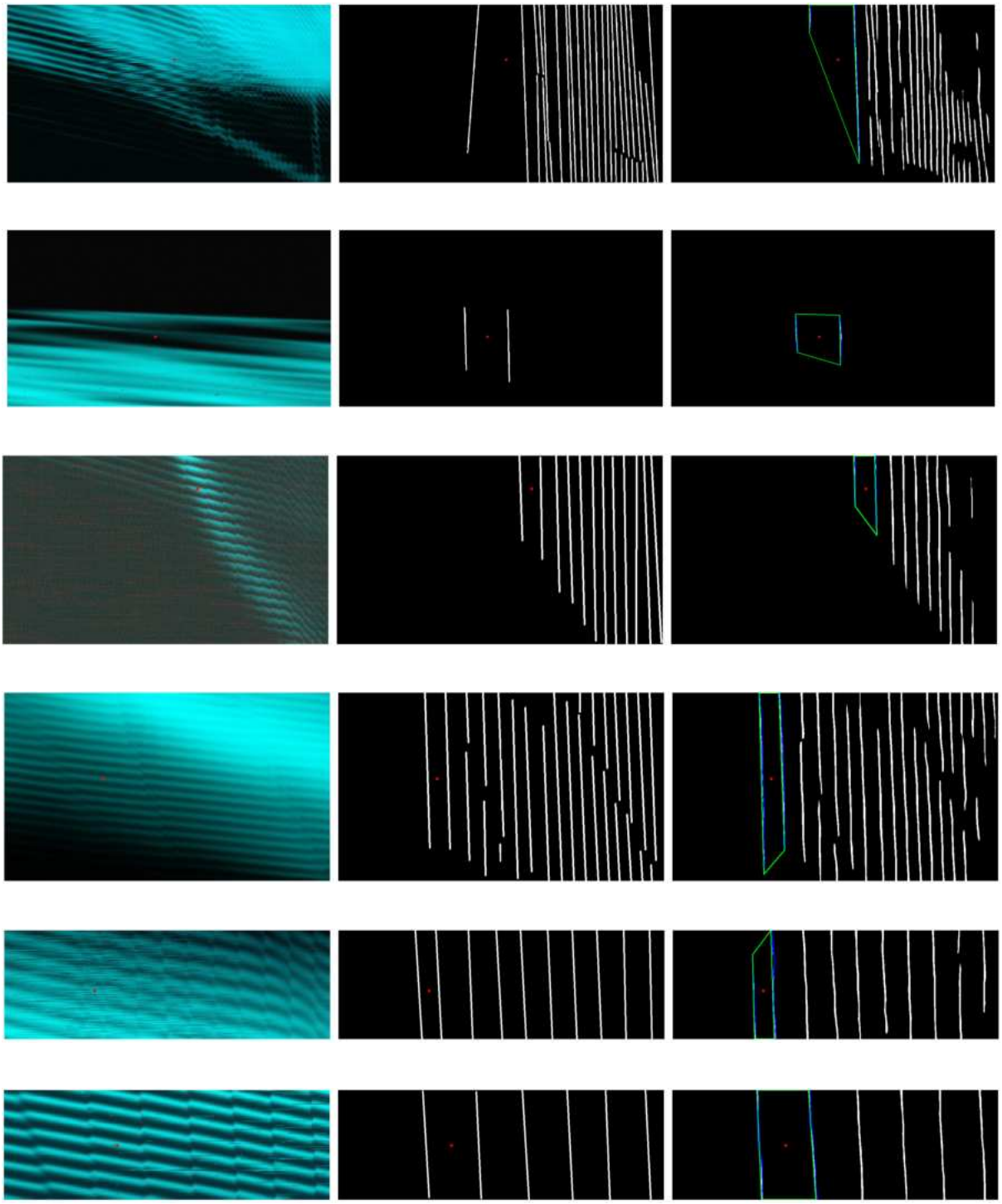


Figure 6.1: Examples of successful single charge regime detection gathered from different devices and showcasing different measurement quality. In the first column we have the original CSD that is used as input to the model, next to it we have the corresponding hand-labeled ground-truth masks. Th third column is the model's output which consist of a binary mask with the predicted transition lines and the localization of the single charge regime. The single charge region is highlighted by a blue and green contour and its center is indicated by a red square patch. The red patch is overlaid on both the stability diagram and the ground-truth mask to demonstrate accurate localization.

Table 6.1: Per-device design aggregate success of detecting the single charge regime

Design	Success (%)	Successful Detection / Total Diagrams
Design A	84%	70/83
Design B	85%	52/61
Design C	79%	64/81
Design D	88%	130/147
Design E	88%	122/138
Design F	85%	121/142
Design G	70%	30/43
Design H	61%	76/124
Design I	75%	147/196
Overall Success	**80.0%**	**812/1015**

The performance varies notably between the different device designs, ranging from 61% (Design H) to 88% (Design D and E) with the majority of devices having a success rate over 80%. Since the number of available samples per design is not uniform (e.g., only 43 samples for Design G compared to 196 for Design I), the confidence in these accuracy estimates differs across devices. The overall pooled accuracy across all devices was 80.0% (812/1015). These results highlight that while the model performs strongly on most devices, performance is lower on a subset (Designs G and H), suggesting potential device-specific challenges or systematic differences in the corresponding data.

Devices with clearer transition lines and higher signal-to-noise ratios tend to achieve higher success rates; conversely, devices with noisy or stochastic lines show reduced performance. This suggests that device-specific measurement conditions are an important factor and motivates the failure analysis in Sec. 6.2. Other reasons for having a low success rate could be defective devices either due to fabrication or design.

Looking at design G we see according to table 4.1 that it was patterned on two different wafers (*ii* and *iii*) from the same batch *B1*. If we look at the results of successfully finding the single electron regime and split them across these two different wafers we see that the 13 failures are split evenly between the two, with 6 failed detections for wafer *i* and 7 for wafer *ii*, therefore we rule out the case that a bad wafer could have resulted in bad devices or measurement in this case as the results are consistent across both. Looking further at the individual failures we see that almost all of them are caused by a spurious line that was detected by the model as a transition, therefore we conclude that the low success rate for design G does not highlight any intrinsic issue with the design nor defective devices from a specific wafer or batch run, but it is due to spurious lines that emerged during measurements and that can be taken care of during online auto-tuning or with the help of additional image processing techniques which we explain in the failure analysis section (see Sec. 6.2). And this conclusion makes more sense when we realize that design G was

tested the least amount of times as it has the lowest number of stability diagrams (43 CSD) than any other device. Therefore we expect that as we perform additional measurements and auto-tuning tests on design G, the success rate would be more representative of the actual numbers.

Regarding design H we also do not find any correlation between the failure rate and a specific batch or wafer, but we do see that the majority of the measurements are of bad quality where they have less defined Coulomb peaks, very faint lines, and large portions of the image where it is totally blank with not much information in it. We associate this to the bad quality of this device which has a differ design structure than the previous ones. We show some examples of the quality of the stability diagrams of design H in Fig. 6.2.

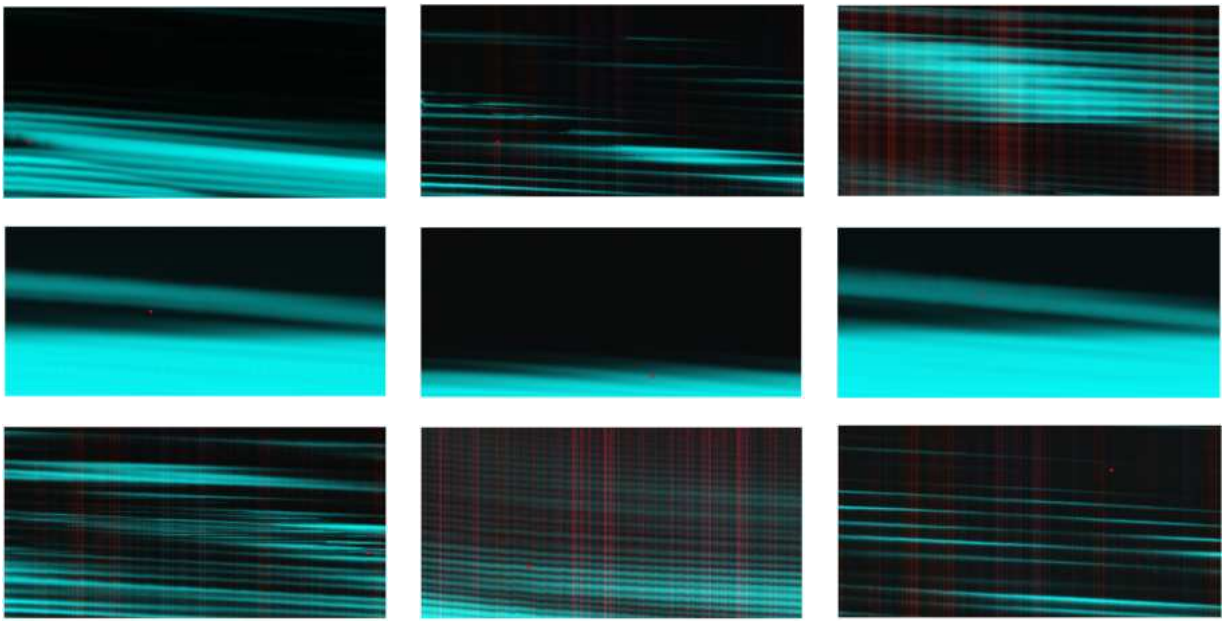


Figure 6.2: Examples of low-quality stability diagrams from device H, highlighting poorly defined Coulomb peaks, very faint or discontinuous transition lines, and large blank regions with little usable signal. These characteristics are consistent with the device design and measurement quality that differ from the other devices and likely explain the elevated failure rate for device H.

Per-Fold Inference Summary

As shown in Table 6.2, the model achieved consistent performance across the five cross-validation folds, with accuracies ranging from 76% to 86%. The mean accuracy over all folds was 80.0% with a standard deviation of 3.6 percentage points, indicating relatively stable generalization performance across different data splits. The results suggest that the model is robust to variations in the training and test partitions, although some variability remains, reflecting the inherent differences between folds as we are grouping on unique devices in terms of architecture, physical die, and gates used.

Table 6.2: Per-fold single charge detection success obtained from 5-fold group cross validation.

Fold	**Correct / Total**	**Accuracy (%)**
Fold 1	174 / 203	86%
Fold 2	162 / 203	80%
Fold 3	154 / 203	76%
Fold 4	161 / 203	79%
Fold 5	161 / 203	79%
Mean ± SD	812 / 1015	80.0% ± 3.6

6.2 Failure Analysis

To further improve the tuning pipeline, we investigate the most common failure modes and their causes. For each failure type we (i) give a concise description, (ii) explain likely root causes, and (iii) propose concrete mitigation strategies and evaluation criteria. We also show examples of these failures and discuss how to adapt both the model and the postprocessing pipeline. We identified five main failure types which we describe in Table 6.3.

Failure Modes	**Associated Observations and Causes**
Bad Quality CSD	• Defective device • Noisy measurement • Large regions with no signal (low SNR)
Missed Lines	• Stochastic lines • Faint lines
Spurious Lines	• Noise-induced false positives • Instrumentation or measurement artefacts
Fragmented Lines	• Low contrast leading to broken lines • Interrupted transitions
Ambiguous Labeling	• Human annotation inconsistencies

Table 6.3: Summary of common failure modes observed in our stability diagrams and associated causes.

Bad Quality CSD

There are multiple reasons for having a bad quality stability diagram. The most common reasons are cases where the QD is of bad quality, either due to fabrication defects or device architecture. There is also the possibility of having a bad SET detector or device

tuning. In a regular fine-tuning procedure, the expert would identify these devices on the fly and they would not be used for fine tuning as they are not considered fit for qubit implementation. One such example is design H which has a big number of defective devices that are not suitable for qubit realization. We show some of its stability diagrams in Fig. 6.2.

In this thesis we did not classify these devices beforehand therefore they were included in our final dataset which ultimately reduced the overall tuning success rate. But in future work, removing these defective devices that will not be used for qubit operation, is the right direction forward and would give a more accurate evaluation of how successful our ML model is at charge auto-tuning and detecting transition lines.

Another instance of a bad quality CSD is when the SNR is low, making it very hard to find the transition lines for both human and machine. There are many reasons for noisy diagram, either due to the measurement itself, a human or machine error during probing, or external environmental factors. Having enough of these samples we can train a neural network to act as classifier of good versus bad devices or stability diagrams, and filtering out all the cases where the measurement should not be used for qubit implementation and therefore would be excluded from the auto-tuning pipeline. We envision developing such a classifier for our online auto-tuning, acting as first filter enabling us to tune the relevant devices only.

Missed Lines

In some instances the model could miss some transition lines or not detect them at all. This can be due to multiple reasons like the ones we listed before (bad quality diagrams or devices) but also in extreme cases of low-contrast CSD where the transition lines are very faint and in some cases where we have stochastic lines which we discussed in Sec. 4.2.1.

For low contrast images we can experiment with different preprocessing techniques like contrast enhancement or even calculating the gradient of the CSD to make the edges more defined, and we can actually stack these on top of the original diagram to be fed to the model as a three channel 'image' to replace our current statistical features channels (see Sec. 4.3.1), or even calculate them from these statistical channels to have a hybrid approach in which we still consider all per-pixel measurements as well as enhancements through contrast and gradient based filters.

As for stochastic lines there are two possible solutions that we can implement. First of all, it is clear that we have a class imbalance when it comes stochastic lines since the majority of transitions are regular lines. Therefore, one solution to mitigate this class imbalance would be to increase the size of the dataset and if possible providing additional measurements which contains stochastic lines in them. Another way to make our model more robust to these lines would be to implement a dynamic line thickness for stochastic

lines in the ground truth mask that would precisely cover the line's width in each case. We currently have a different thickness in our generated ground truth masks for regular transition and stochastic lines, but this does not take into account the wide variety in width of different stochastic lines. Therefore, the best way to achieve that is by re-labeling the specific diagrams that contains stochastic transitions with an appropriate thickness that is chosen on a per-line basis.

We show in Fig. 6.3 a case where the first transition line has a very high degree of stochasticity which caused a failure in detecting the single charge regime.

Spurious Lines

Spurious lines are sometimes detected as actual transition lines by our model. This is one instance of a false positive detection. It is not a surprise that the model could not easily identify the difference between a spurious and a transition line as there is almost no difference between the two when it comes to the shape of the line or its features (unlike stochastic versus transition lines). The biggest differentiator between the two, is that sometimes spurious lines could have a different slope than the rest of the transitions, or it could have a wider gap when compared to the regularly spaced transition lines but it is not always the case and therefore it is tricky to tell exactly what consist a spurious or a transition line, although our model was successful in ignoring spurious lines in multiple instances. We associate this success to our full diagram implementation of the line detection where the model was able to have a holistic view of the stability diagram and therefore make connections related to not only the features of a transition lines but also its characteristics like slope, direction and spacing which would add relevant information to the model and make more accurate decisions.

We show in Fig. 6.4 and example of a CSD which contains a spurious line which was wrongfully detected by our model. Note the slight shift in the slope of the spurious line when compared to the other lines. We also have multiple instances where our model accurately ignored spurious lines, and was successful in detecting the single charge regime (see Fig. 6.5).

Fortunately there are methods we proposed to mitigate this issue. One of them is using a fast Fourier transform to convert the predicted binary diagram from the image space to the frequency domain, in which we would clearly be able to identify spurious lines based on their tilt or different orientation when compare to other transition lines, and also based on the spacial frequency denoting the separation of the regularly spaced transition lines.

Moreover, we can easily take care of spurious detections during online tuning, at the cost of performing an additional measurement and feeding both measurements to the model, therefore locating the change in position that such line would normally exhibit

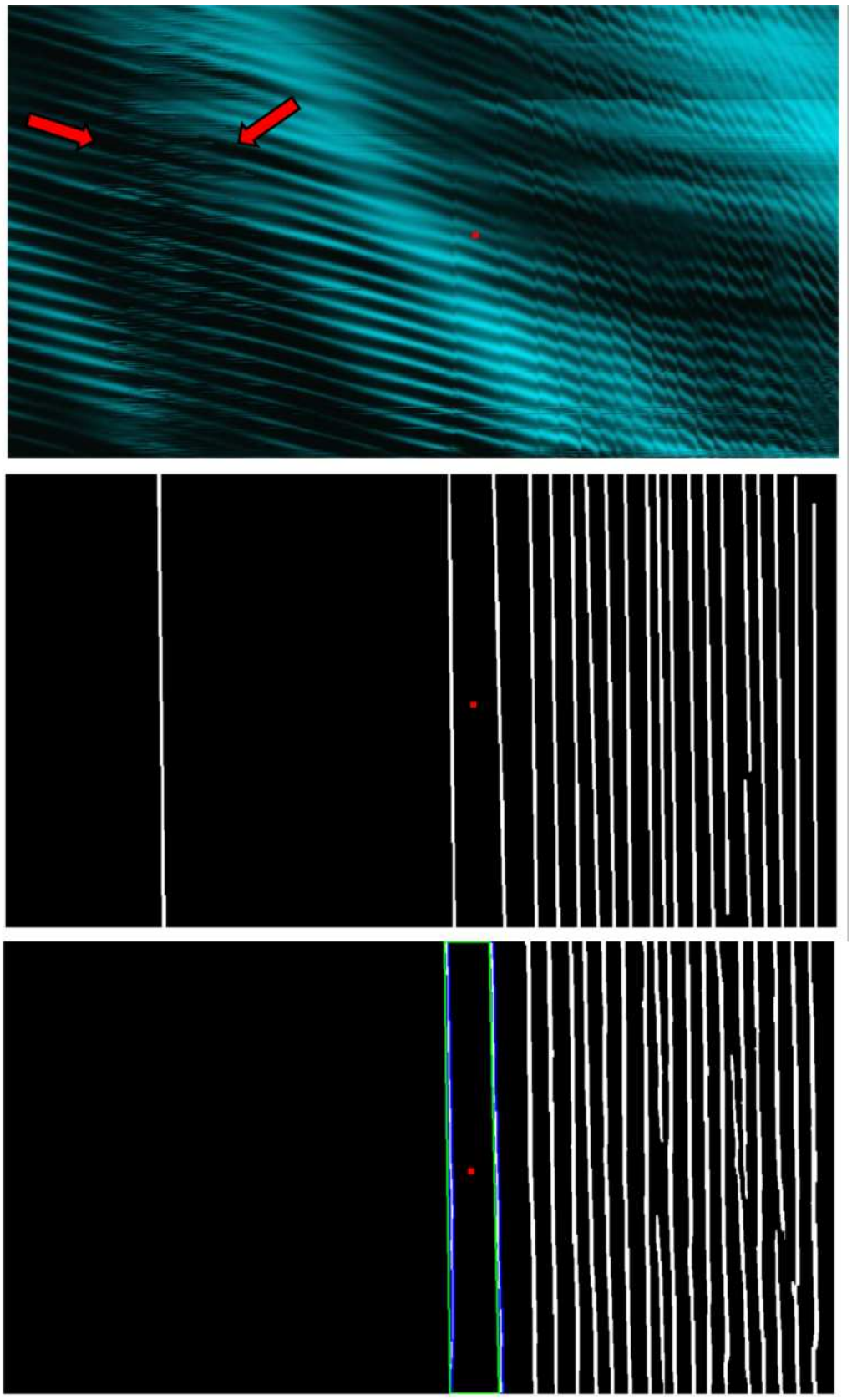


Figure 6.3: Red arrows pointing to a stochastic line in the input CSD (first image) which was labeled as a transition line in the ground truth mask (second image) but was not detected by our model (third image).

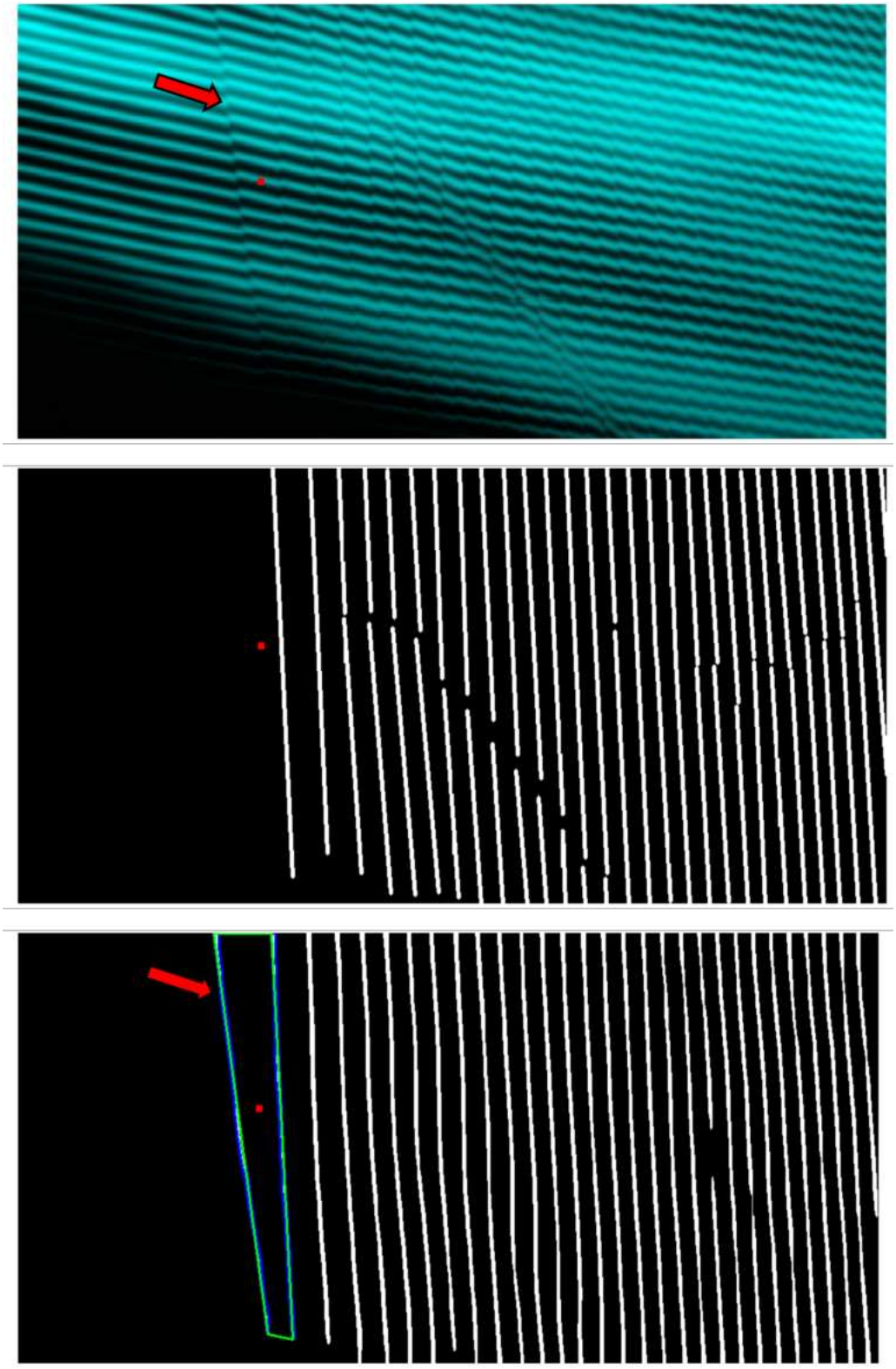


Figure 6.4: Example of a false detection due to a spurious line. From top to bottom we show the input CSD, the labeled ground truth mask, and the faulty prediction, with the red arrow pointing at the spurious line which was wrongfully predicted as the first transition by our model.

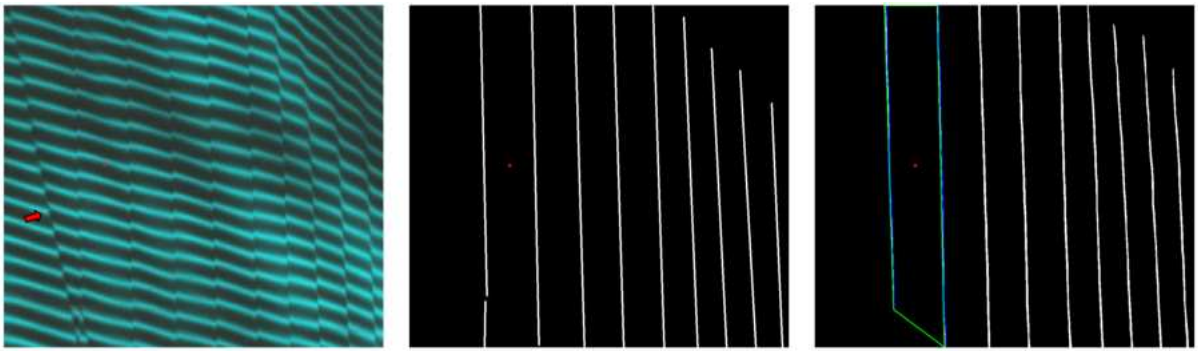


Figure 6.5: Accurate detection of the single electron regime in the presence of a spurious line before the first transition (the red arrow is pointing at the spurious line).

and then we can label it as spurious. In addition to multiple CSD measurements we can also re-scan in different voltage ranges.

Fragmented Lines

Sometimes the transition lines are broken in the prediction mask but also in the original CSD measurement. This rarely causes issues in detecting these fragments, but would sometimes cause issue in the postprocessing steps that would attempt to find the single charge regime from the predicted transitions. As we discussed in Sec. 5.4, our postprocessing loop which take place after inference consists of performing multiple classical image processing steps and one of them is locating the first two transition lines as being the boundary of the single electron (or hole) regime. One issue arise in cases where the first transition line is broken in two parts, the hard-coded postprocessing technique would fail due to misclassifying these two fragments-belonging to the same transition lines-as being completely unique transitions and thus labeling them as the first and second transition. We show such an example in Fig 6.6 where the first transition line is predicted not as a continuous line, but as three broken segments of that line (mainly because that line is stochastic), and thus our postprocessing method could not identify these three segments as being part of the same line, because of the slope which makes it ambiguous to tell the difference between the segments belonging to the same transition or forming unique transitions on their own.

Our postprocessing technique works relatively well for broken segment that are not part of a tilted line, or when the slope of that line is not steep, but when the original transition line is not straight and our model did not detect it in its entirety due to some noise artifacts or stochastic features, then it becomes hard to define what fragments belong to the first transition and which ones belong to the second or third. One way we can mitigate this problem is by thinking of what consist a transition and following a logic analogous to what is done in a lab when locating say the nth electron regime. We

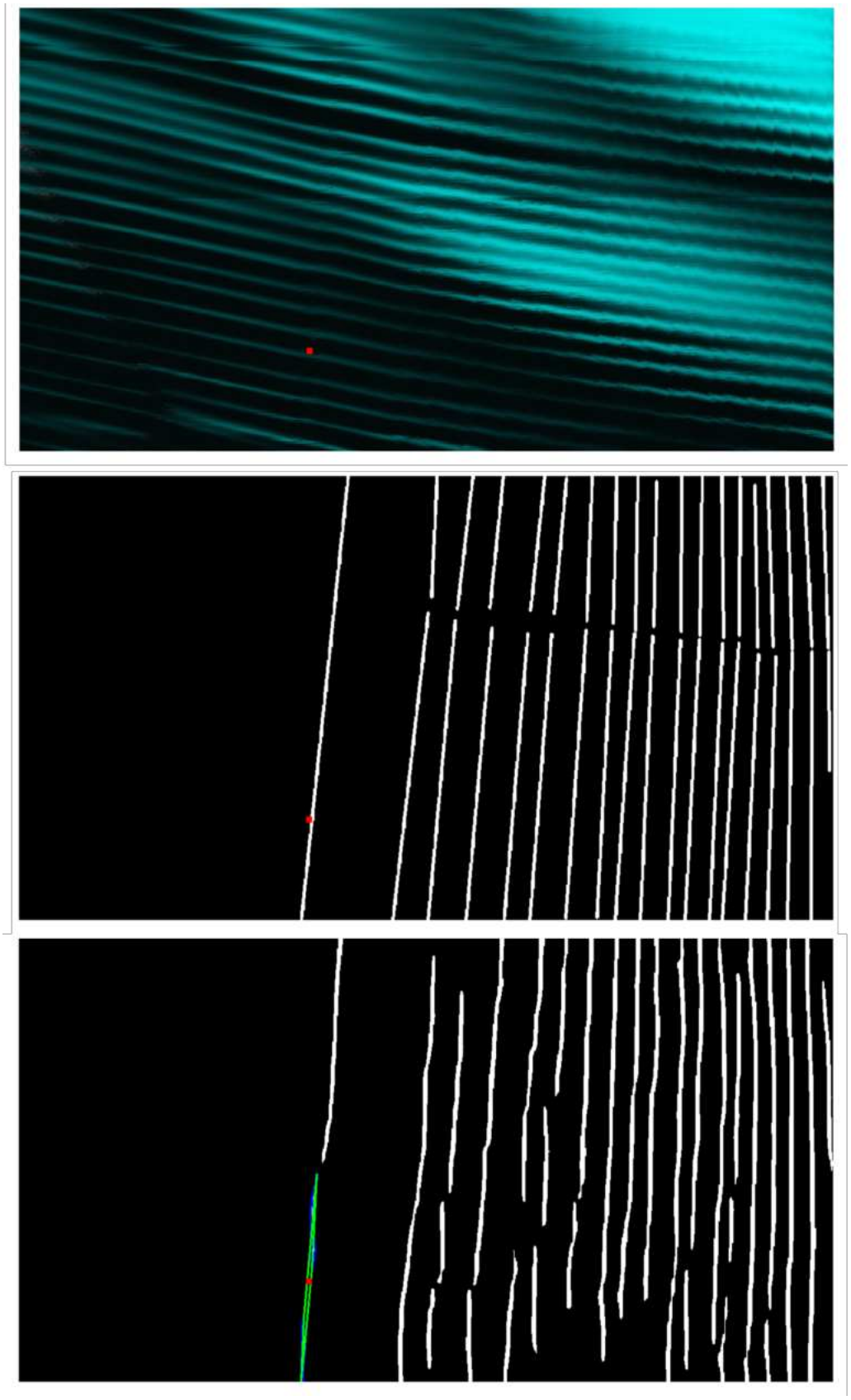


Figure 6.6: Example of a faulty detection due to a fragmented transition line in the prediction mask.

can make our postprocessing script sweep along the x axis individual rows and when it encounters a set of white pixels (representing a transition line) it adds 'plus one' to a counter, and after locating the set of fragmented lines that delimit the first and second transitions we identify the area between them as being the single charge regime. In this way we can find the first transition across different locations on the y-axis, but this method would fail in special cases where the CSD measurement is out of bound and the transition lines are not fully covering the diagram, therefore a more robust technique needs to be developed.

Mislabeling or Ambiguous Labels

Some cases are genuinely ambiguous. An expert, labeling the diagrams may be uncertain whether a line is an actual transition or a spurious line, or if two very close lines are actually unique transitions or some measurement artifact causing the emergence of dual lines with some degree of stochasticity between the two. For such cases, the experimentalist have the ability to perform additional operations during an online tuning procedure, like zooming in in the area of interest to remove any doubts about the nature of the lines, or repeat the measurements under slightly different conditions etc. This level of flexibility was not available for us during this offline study performed on static stability diagrams, but in an upcoming online tuning experiment we expect to remove this ambiguity by having the freedom to perform additional operations as required. We also envision mapping the logic of what operations the experimentalist might need to perform to a set of algorithms that would capture the logic behind this workflow and give intelligent feedback about what needs to be done to increase the confidence of the model and remove any uncertainty in a fully autonomous way. In addition to that, we offer the below possible solutions to remove any ambiguity on the labeling level of static CSDs.

- **Annotation protocol:** use multiple annotators and resolve disagreements with consensus or a senior annotator; include an "ambiguous" flag in the label set.

- **Active learning:** let the model propose uncertain regions for human review (model confidence vs label mismatch).

- **Online verification:** where feasible, perform in-situ checks (repeat measurements, gate scans) to resolve ambiguity; use these confirmed cases to correct the offline labels.

6.3 Results summary

This chapter presented a comprehensive evaluation of the offline auto-tuning pipeline. In brief: across the full dataset of 1,015 Coulomb stability diagrams the proposed segmentation plus localization pipeline achieved a pooled success rate of **80.0**% (812/1015). Per-design performance varies substantially (61%–88%, see Table 6.1) and per-fold accuracy is stable (mean 80.0%, SD 3.6%, see Table 6.2). Qualitative inspection (e.g. Fig. 6.1 and Appendix B) confirms that the system reliably localizes the single-charge regime in many realistic cases, while exposing a small number of systematic failure modes (Sec. 6.2).

Chapter 7

Discussion and Future Work

In this chapter, we outline several concrete directions to extend and strengthen our ML-based auto-tuning framework. We begin by describing real time, in prober deployment, then discuss on-chip implementation, scalability to multi-qubit arrays, richer feature extraction, and enhanced input representations.

7.1 In-Situ Auto-Tuning in a Cryogenic Wafer Prober

Integrating our segmentation model directly into the cryogenic prober control loop is the most immediate next step. The online deployment consists of loading the trained CNN onto the control unit of the cryoprober, hence achieving low latency inference with a closed-loop voltage control. This hardware integration is especially interesting as it gives us the ability to benchmark live tuning throughput, success rate, and operator effort reduction. In addition to that we can incorporate on-the-fly repeat scans to mitigate spurious lines detection and noise fluctuations, something we could not do with offline tuning on static CSDs. For instance, we can automatically re-scan regions where confidence is low or spurious lines appear, based on a feedback loop built into the model, hence reducing false positives due to noise and other unpredictable factors. In order to achieve that, additional work must be done to develop this feedback mechanism into our the model and to make decisions on what operation should the measurement apparatus do, like zooming in on a specific area of interest, or repeat the measurement due to the detection of noise etc.

Concretely, the online pipeline would operate as follows:

1. **Continuous Data Stream:** As the measurement system sweeps (V_{G1}, V_{G2}), the acquired currents are streamed directly into GPU memory, bypassing the need to save static stability diagrams.
2. **Real-Time Preprocessing:** Normalize, resize, and compute statistical maps (mean, median, std.) on the fly.

3. **Immediate Segmentation:** Run the U-Net encoder–decoder to produce a binary line mask.
4. **Voltage Update:** Compute the centroid (V_{G1}^*, V_{G2}^*) of the area enclosed between the first two transition lines and apply these voltages to reposition the device in the single-charge regime.
5. **Feedback Assessment:** Compute a model confidence score. If below a threshold, trigger a high-resolution sub-scan (zoom) and re-infer.

By closing the loop in situ, we can dramatically shorten tuning cycles, autonomously compensate for noise, and support rapid initialization of a large number of QD devices.

7.2 On-Chip Cryogenic Implementation

Looking further ahead, one goal would be to embed the auto-tuning algorithm inside the dilution refrigerator itself. Key challenges and considerations include:

- **Cryo-Compatible Electronics:** Researchers are working on developing low-power electronics that can operate at sub-kelvin temperatures which would be used as a control unit for instance to run the auto-tuning algorithm. From the model architecture's point of view, it would be necessary to build a lightweight and efficient algorithm that would run in such a restricted hardware.
- **Ultra-Low Latency:** Minimize signal path lengths by placing inference hardware as close as possible to the quantum chip.
- **Thermal Budget and Mounting:** Balance electronic heat dissipation against refrigerator cooling power.

Such on-chip deployment would enable real-time qubit initialization without compromising coherence or adding significant wiring overhead.

7.3 Scalability to Multi-Qubit Arrays and New Materials

Our ML pipeline naturally extends to larger two-dimensional QD arrays and alternative material platforms (e.g., Ge/SiGe, GaAs). All we need is a few examples of labeled CSDs from a new material or device geometry, and by leveraging transfer learning we can fine tune our pretrained model on these diagrams, thus extending the applicability of our model to new devices and accelerating deployment with minimal annotation effort. Note that on top of fine tuning, additional postprocessing techniques need to be configured for devices formed in a multi-qubit array as opposed to single-gate, in order to successfully locate the region of interest.

7.4 Physics-Guided Feature Extraction

Our full-diagram segmentation produces a precise digital map of every charge-transition line in a CSD. Beyond locating the single-charge regime, these line maps enable:

- **Multi-Charge State Identification:** Simply select the kth line pair to target the k-electron (or hole) regime without retraining or re-running inference.
- **Capacitance and Coupling Analysis:** Compute the slopes and spacings of detected lines to extract lever arms, inter-dot capacitances, and cross-couplings.
- **Device Quality Metrics:** Track statistical distributions of device quality across wafers to inform fabrication process improvements.

Given that our model detects all transition lines in a CSD, we can use this to extract more information than just the single charge regime. We can decide to locate the regime of any number of charges by just specifying what region we are interested in since this is done in the postprocessing step therefore there is no need to repeat the inference or create a new prediction. We only need to run inference ounce and we have all the information we need, and we know the location of all the transition lines predicted by our model. In contrast, the patch-based approach that some groups have employed, would require a new run of the algorithm with their ML line classification inference, if they decide to for example locate the two electron regime as opposed to the single electron region.

Moreover, we can extract additional information from this digital map during post-processing, like the slope of the lines, and the distance between them, all of which have real physical meanings that can be very valuable to the team that design and fabricate these devices, helping them push forward their quality and reproducibility. For example in Fig. 7.1, we show how we can extract from the predicted binary mask, the gate voltage value corresponding to the first transition line from its intersection with the horizontal line extending from the center of the y-axis which corresponds to the SET applied voltages. This physical feature, as well as others like the separation between transition lines (drawn in yellow in the same figure) contains very important information about the behavior of the device and its physics. All of this is possible due to our choice of implementing a full-diagram line detection which we proposed in Sec. 3.4 and built our model around it.

7.5 Enhanced Multi-Channel Input Representations

Finally, we can enrich our input beyond the existing three statistical channels (mean, median, standard deviation) by incorporating:

- **Gradient Channels:** Sobel derivatives to emphasize linear features and sharpen transition boundaries.

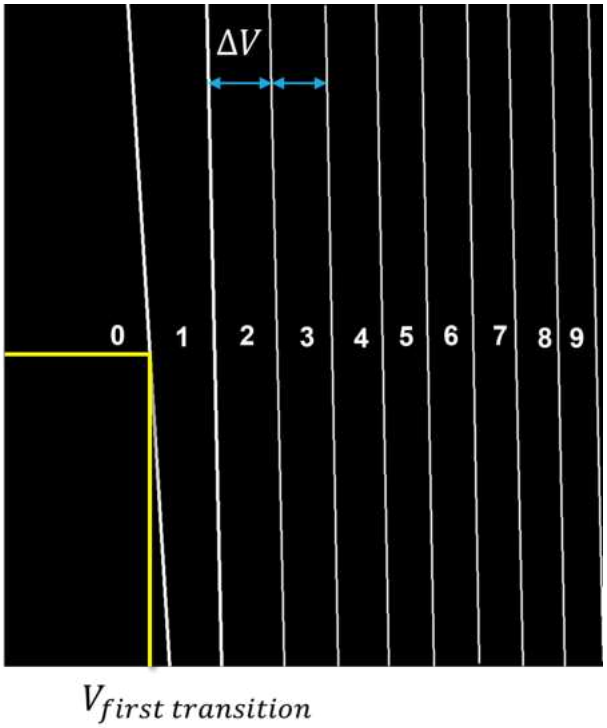


Figure 7.1: Schematic of a binary image containing the predicted transition lines in white, showing the extraction of possible geometric features which holds relevant physics information. In blue we have the separation distance between transition lines which can be mapped to a voltage value ΔV, and in yellow we show the extraction of the of gate voltage $V_{first\,transition}$ (on the x-axis) corresponding to the first transition line. The numbers denote how many charged particles each region would trap.

- **Frequency-Domain Features:** Local Fourier transforms to capture periodic noise patterns and discriminate true transition lines from spurious ones.
- **Auxiliary Physical Signals:** Simultaneously feed in derivatives of current with respect to voltage (e.g., dI/dV_{G1}), adding direct sensitivity to transition thresholds.

These richer representations can be stacked into multi-channel tensors and are likely to boost both detection accuracy and robustness under challenging measurement conditions.

In summary, the path forward combines real-time in-prober deployment, physics-informed post-processing, cryogenic on-chip integration, scalable array support, and richer data representations. Together, these developments will push us closer to fully autonomous, high-throughput tuning of large-scale spin-qubit QD devices.

Chapter 8

Conclusion and Perspectives

In order for any qubit platform to achieve the full promise of quantum computers these systems must first solve the daunting challenge of scalability. This would look different for each qubit platform. In the case of spin qubits realized in laterally-defined QDs, there are two main obstacles to overcome. The first one i) is the ability to fabricate high quality semiconductor QDs which are reproducible and have a low amount of variability. The second hurdle to scalability is ii) having the ability to tune these devices to the desired mode of operation. Tuning is currently being done manually by experts, which makes it a laborious, time consuming process that is prone to errors. Indeed, we have discussed at length the tuning process, and the importance of automating it as we increase the number of devices and their complexity.

This thesis is a direct attempt to solving the second problem, more specifically to automate the charge tuning process which consists of trapping the desired number of charges in each individual QD. Although not a direct objective of this thesis, we also provided in this work valuable feedback towards making informed decision that could help improve the fabrication and reproducibility of these devices, thus making our work directly relevant to both challenges on the route to scaling QD-based quantum computers.

The landscape of automated charge tuning for QDs is characterized by a clear evolution from classical, heuristic-driven methods to sophisticated ML approaches. Early classical algorithms, such as those by Lapointe-Major and Baart, demonstrated the fundamental feasibility of automated line detection and charge state identification, often relying on image processing techniques or predefined patterns. While effective for their specific contexts, these methods typically suffer from limited generalizability, requiring significant prior device knowledge or being restricted to particular device architectures. This dependency on device-specific heuristics presents a significant hurdle for scaling, as each new device or architecture necessitates substantial manual recalibration. The advent of ML has offered a powerful avenue to overcome these limitations. ML algorithms, particularly CNNs, are very good at learning complex patterns from data, enabling more

robust and generalizable solutions. The "empty-then-reload" strategy has emerged as a universal, physically intuitive framework for charge tuning, providing a robust pathway from an unknown charge state to a desired configuration by establishing a known reference point. The challenge for ML methods within this framework lies in efficiently and reliably executing each phase.

Our proposed solution, employing a CNN for full-image scanning of CSDs to identify all transition lines and directly extract voltage values, represents a distinct approach. Compared to prevalent patch-based methods, this full-image analysis offers the advantage of a holistic view, allowing the CNN to leverage broader spatial relationships and contextual information. This could lead to more robust detection of transition lines, even in the presence of localized noise or minor imperfections, potentially mitigating issues like interruptions in transition lines that plague patch-based reloading. The ability to directly identify the center of the desired regime combines detection and localization in a single process, which is also a clear departure from the empty then reload approach widely employed in the industry.

We summarize the key contributions of this work as follow:

Novel Auto-Tuning Pipeline: We have developed a comprehensive full diagram ML-driven framework for the automatic tuning of gate-defined silicon QDs. By leveraging a CNN model built in a UNet architecture on a pretrained MobileNetV2 backbone for semantic segmentation of charge transition lines in stability diagrams.

Quantitative Validation: Through this model, we have achieved robust offline tuning performance across diverse device designs and fabrication conditions by demonstrating average per-device accuracies exceeding 80%, highlighting pathways to approaching near-perfect performance on good quality devices.

Comprehensive Dataset: Curated and manually labeled over 1,000 CSDs from nine distinct device geometries, two carrier polarities, four fabrication runs, and seven wafers. This diversity exceeds that of previous studies—e.g., Yon *et al.* used $\sim$27 diagrams split between three device types for their offline tuning [93]. Our dataset captures real-world variability and underpins the model's generalizability.

Achieved Semi-Supervised Labeling: The model we developed in this work was used to label new stability diagrams, due to its superior speed and detection accuracy over manual labeling. By deploying our trained network as an accelerated annotator, we achieved semi-supervised labeling of newly acquired CSDs, reducing human effort compared to fully manual annotation.

Physics-Informed Post-Processing: Beyond locating the single-charge regime, our full-diagram segmentation yields digital line maps from which capacitance, lever arms, inter-dot couplings, and line slopes can be extracted. This physics-guided feature extraction provides actionable feedback to device fabrication teams, an advantage not afforded by patch classifiers.

Taken together, these contributions mark a significant advance toward fully automated, closed-loop charge tuning in QD systems. By shifting from heuristic, manual procedures to data-driven semantic segmentation, we reduce the calibration overhead and improve reproducibility, which is a key requirement on the road to achieving fault-tolerant quantum computers.

Nonetheless, several challenges remain. First, real-time, low-latency inference on resource-constrained hardware demands further model compression and quantization studies. Second, device drift and variability call for self-supervised or active-learning paradigms to adapt models on the fly with minimal human labeling.

Looking forward, we envision three near-term objectives. (1) Deploy the auto-tuning pipeline in a production cryoprober and benchmark live tuning throughput and accuracy. (2) Extend the dataset to different designs and material stacks and possibly to larger QD arrays. (3) Extract physics-based features from the predicted stability diagrams to further aid in understanding and developing these devices.

By bridging the gap between advanced deep learning methods and the demands of quantum hardware, this work lays the foundation for the next generation of automated qubit calibration. As QD devices grow in complexity and number, our data-driven auto-tuning framework will be a critical enabler of rapid, reliable, and scalable quantum information processing.

Appendix A

Training Details and Metrics

This appendix documents the exact training configuration used in the experiments reported in Chapter 6, describes the grouped cross-validation protocol that ensured per-device separation between training and test partitions, provides training metrics and diagnostics for a representative fold, lists the principal software components and their licenses, and discusses practical limitations and interpretation.

A.1 Hyperparameters

All folds were trained with the same hyperparameter set (no per-fold hyperparameter search or tuning was performed). The table below lists the hyperparameters that were kept fixed during the experiments.

Table A.1: Fixed hyperparameters used for all folds.

Hyperparameter	**Value**
Optimizer	Adam [71]
Learning rate	1×10^{-4}
Loss function	Dice loss
Batch size	8
Epochs	50
Input size	1024x1024
MobileNetV2 width multiplier	$\alpha = 1.4$

The values above are the parameters that remained unchanged across the 5 folds. Implementation details (preprocessing, encoder/decoder choices and how the encoder was initialised and trained) are described in the next section.

A.2 Architecture, preprocessing and trainability

A.2.1 Architecture Overview

The model pairs a MobileNetV2 [98] encoder ($\alpha = 1.4$) with a small custom decoder. In the experiments described here the encoder was initialized from ImageNet weights and *left trainable* (i.e., the encoder was fine-tuned during training).

Why MobileNetV2

MobileNetV2 was adopted as the encoder due to its favorable accuracy–efficiency tradeoff. The key architectural benefits are listed below with brief explanations.

- **Depthwise separable convolutions**
- **Pointwise convolutions**
- **Inverted residuals**
- **Linear bottlenecks**

The separable convolutions in MobileNet split a standard $k \times k$ convolution into a *depthwise* spatial convolution (per-channel $k \times k$) followed by a *pointwise* 1×1 convolution for channel mixing. This decoupling greatly reduces parameter count while preserving effective receptive field.

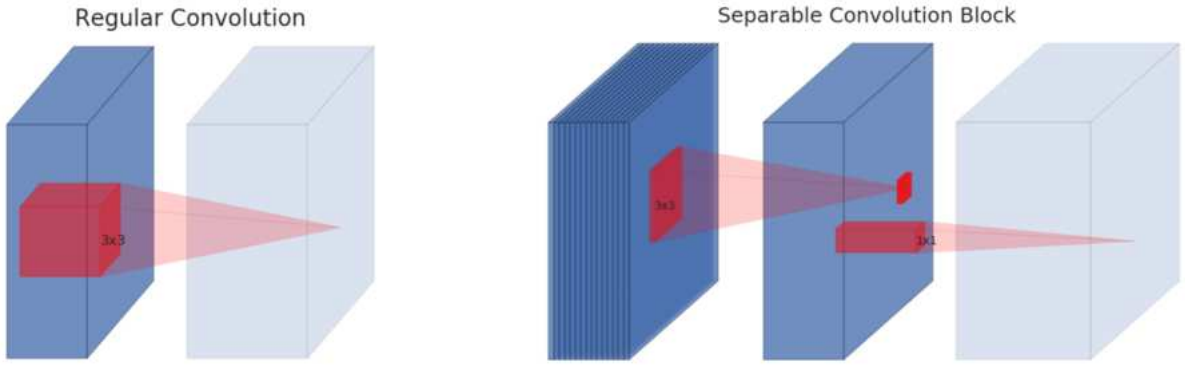


Figure A.1: Schematic showing the difference between the regular and separable convolution block which is implemented in the MobileNet architecture (Adapted from [98]).

A.2.2 Preprocessing

- Input images were resized and zero-padded (as needed) to a fixed training resolution of 1024×1024 prior to augmentation and batching.

- Masks and probability maps are inverse-resized and unpadded in post-processing before task-level evaluation so we detect the single charge regime with the original pixel location.

A.2.3 Model Size and Parameter Count

The segmentation model used in this work contains a total of **2,057,489** parameters (approximately **7.85 MB** when stored as 32-bit floats). Of those, **2,032,497** parameters are trainable and **24,992** are non-trainable. The small number of non-trainable parameters indicates that only minor quantities (e.g. BatchNorm) were non-trainable, and that the MobileNetV2 encoder was fine-tuned together with the decoder during training (hence nearly all parameters were updated by gradient descent).

For context, this model ($\approx$ 2.06M parameters) is compact compared with many full U-Net variants used in other segmentation tasks (which commonly have an order of magnitude more parameters). The modest parameter count reduces memory and storage requirements and makes the model practical for repeated training and inference in the experimental pipeline used in this work and ideal for online deployment.

A.3 Cross-Validation Protocol and Dataset Split

- **Group splitting:** we used grouped k-fold cross-validation with $k = 5$ where the grouping key corresponds to the unique physical device identifier (device design + die location + gates used). This ensures that all images from a single physical device are confined to a single fold and cannot appear in both training and test partitions for the same fold, therefore no data leakage takes place.

- **Fold proportions:** within each fold the split is approximately 80% (training set) and 20% (test set) of images. Each fold therefore provides a held-out test set drawn from devices that the model did not see during training for that fold.

- **Use of test partitions:** we did not use the fold test sets to tune hyperparameters. The fixed hyperparameter set (Table A.1) was used for every fold. The grouped folds were used to obtain robustness estimates across devices, not for hyperparameter selection.

- **Why k-fold** Using k folds allows us to (i) use all labeled data for both training and held-out evaluation across different folds and (ii) report metrics over folds to capture dataset heterogeneity.

A.4 Training Metrics and Interpretation

Below we summarize the behavior observed in one of the folds. A multi-metric plot of train vs test curves across epochs for loss, Dice coefficient, Precision, Recall and Intersection over Union (IoU) (see Figure A.2).

A.4.1 Metrics Summary

- **End-of-training snapshot:** by the final epoch the training Dice is high (≈ 0.84) while the validation Dice is lower (≈ 0.59). Validation Precision stabilizes higher than validation Recall (≈ 0.70 vs ≈ 0.56 in the representative fold).

- **Convergence:** training loss and training Dice improve steadily during the 50 epochs; validation metrics improve too but show a persistent train/validation gap that plateau toward the end of training.

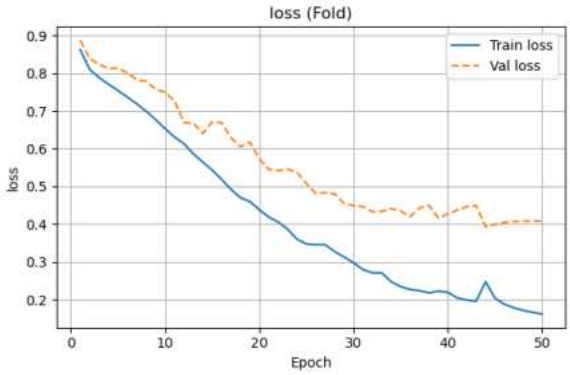
- **Precision–Recall trade-off:** across epochs validation Precision stabilizes at a higher value than validation Recall in the representative fold, which is consistent with a model that returns fewer false positives but can miss portions of the transitions or predict a skeleton version of the thick line.

A.4.2 Interpretation

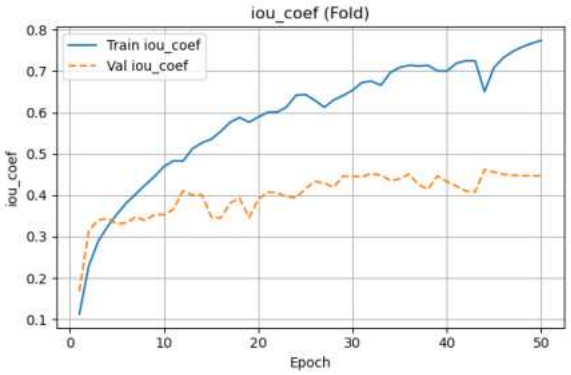
The main factors that explain the numerical behavior above are the following.

Mask geometry vs. practical task A core observation is that pixel-wise segmentation metrics (Dice, IoU) are strongly affected by the geometry of the ground-truth masks. In our dataset the annotated transition lines were often drawn with relatively thick contours. The trained model tends to predict thinner, well-localized lines along the location of the transition lines seen in the stability diagram. A thin, correctly localized prediction therefore yields a relatively low pixel-wise overlap with a hand labeled ground-truth having thick lines, while still being sufficient for the downstream single charge detection task. In short *A correctly localized thin predicted line may yield low pixel-wise overlap with a thick ground-truth mask thus yielding smaller values for IoU, DICE, etc.*

Post-processing influence Postprocessing steps (morphological operations, connecting broken lines, spurious-component filtering) materially affect the task specific success. These steps can improve the detection of continuous transition lines even when raw pixel-wise metrics remain modest. Therefore, reporting a success rate of finding the single charge regime is more meaningful in our case than reporting pixel-wise metrics.

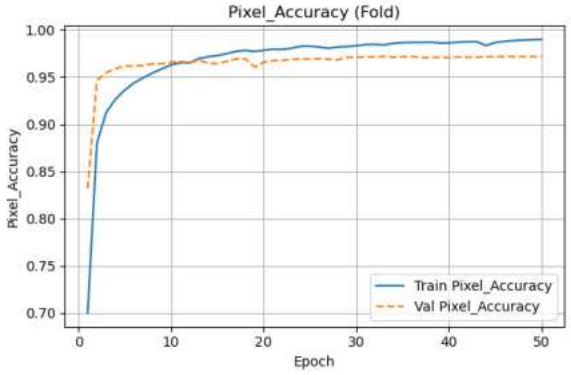


(a) Loss

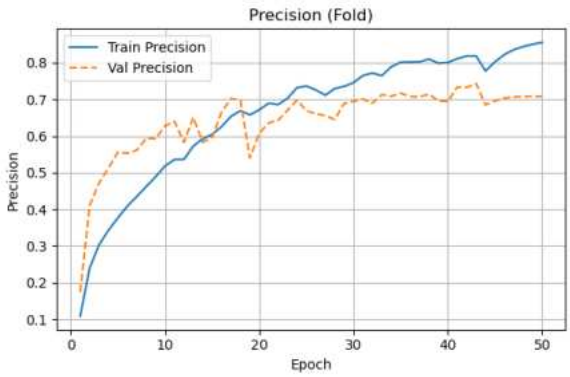


(b) Mean IoU

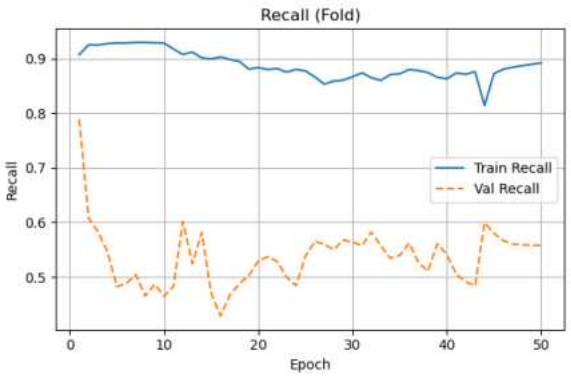
(c) Dice coefficient



(d) Pixel Accuracy



(e) Precision



(f) Recall

Figure A.2: Training and evaluation metrics for the segmentation model. From left-to-right, top row: (a) Loss, (b) Mean IoU. Middle row: (c) Dice coefficient, (d) Pixel Accuracy. Bottom row: (e) Precision, (f) Recall.

Dataset heterogeneity and device-specific patterns Some device designs are under-represented. The network can overfit to abundant patterns in the training subset causing poorer generalization and lower test performance on those under-represented device types.

A.5 Labeling issues and their effect on metrics

Manual annotation is one of the most important source of label variation in our dataset and a dominant contributor to lower pixel-based metrics. Hand labeling of transition lines is inherently noisy with inter annotator variability. When multiple human annotators label a dataset, agreement may be imperfect. We highlight the practical points below.

Annotator style and mask thickness Ground truth masks typically had thicker lines in order to make sure they fully cover the features of the actual transitions in the CSD and to prevent having class imbalance with so few thin lines in a background of black pixels. This conservative drawing of transition lines introduces a systematic mismatch: when the model predicts a thinner, accurately placed line, the overlap with the thick ground-truth contour can be small even though the predicted line is experimentally useful for locating the single charge regime. This thickness mismatch is one of the principal reasons for reduced Dice and IoU values in the experimental (hand-labeled) dataset.

Effects of resizing and interpolation Preprocessing resizes and pads raw images to the training resolution. Resizing alters the effective line thickness in the mask. These geometric artifacts further reduce pixel overlap between prediction and ground truth.

Inter-annotator variability and label noise When multiple humans annotate the same image they do not always agree on boundary placement or thickness. This inter-annotator variability places a practical upper bound on achievable pixel-wise metrics.

A.6 Software and License Statement

The code and experiments reported in this thesis use open source software and pre-trained weights. The encoder used in the U-Net model is `MobileNetV2` accessed via `tensorflow.keras.applications` with the pre-trained ImageNet weights. All model implementations and weights are released under the Apache License, Version 2.0. Other libraries used include TensorFlow/Keras, NumPy, pandas, OpenCV, matplotlib, seaborn and scikit-learn (all permissively licensed).

Appendix B

Offline Auto-Tuning Test Results

This appendix presents the detailed results of the offline auto-tuning test and collects additional examples of model inferences across different devices and measurement qualities. The examples were selected to illustrate the variety of signal/noise conditions encountered in our dataset and the model's robustness under these conditions. For the legend and symbol meanings (contours, red patch, etc.) see Figure 6.1 in the main text. These qualitative examples complement the quantitative per-device design and per-fold results presented in Tabled B.1.

Table B.1: **Inference Summary per Design and per Fold.** For each device and fold we report the ratio of the number of diagrams with successful single charge detection to the total number of diagrams, followed by the percentage value.

Fold	Design A	Design B	Design C	Design D	Design E
1	11/12 (92%)	14/16 (88%)	17/17 (100%)	39/42 (93%)	24/24 (100%)
2	17/19 (89%)	10/11 (91%)	15/20 (75%)	21/26 (81%)	37/41 (90%)
3	11/15 (73%)	7/12 (58%)	12/15 (80%)	25/30 (83%)	25/31 (81%)
4	12/14 (86%)	9/9 (100%)	13/15 (87%)	21/25 (84%)	17/21 (81%)
5	19/23 (83%)	12/13 (92%)	7/14 (50%)	24/24 (100%)	19/21 (90%)
Total	70/83 (84%)	52/61 (85%)	64/81 (79%)	130/147 (88%)	122/138 (88%)

Fold	Design F	Design G	Design H	Design I
1	17/18 (94%)	5/8 (62%)	19/29 (66%)	28/37 (76%)
2	23/24 (96%)	6/6 (100%)	8/20 (40%)	25/36 (69%)
3	22/28 (79%)	5/9 (56%)	15/26 (58%)	32/37 (86%)
4	43/46 (93%)	6/9 (67%)	12/22 (55%)	28/42 (67%)
5	16/26 (62%)	8/11 (73%)	22/27 (81%)	34/44 (77%)
Total	121/142 (85%)	30/43 (70%)	76/124 (61%)	147/196 (75%)

Grand total (all shown devices): 812/1015 (80.0%)

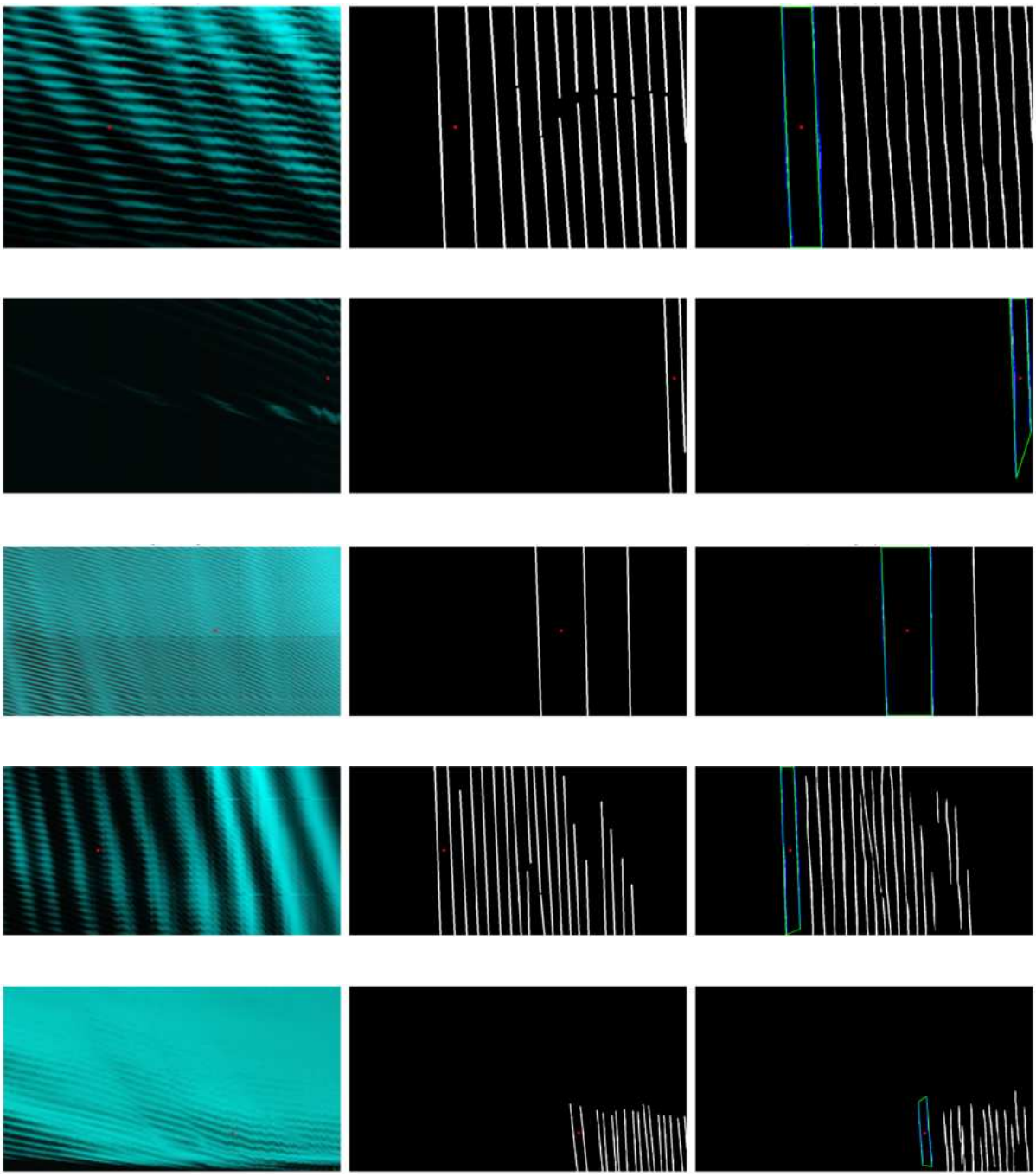


Figure B.1: Additional successful inference examples (part A). Each panel shows the original stability diagram, the hand-labeled ground-truth mask, and the model prediction (legend as in Figure 6.1).

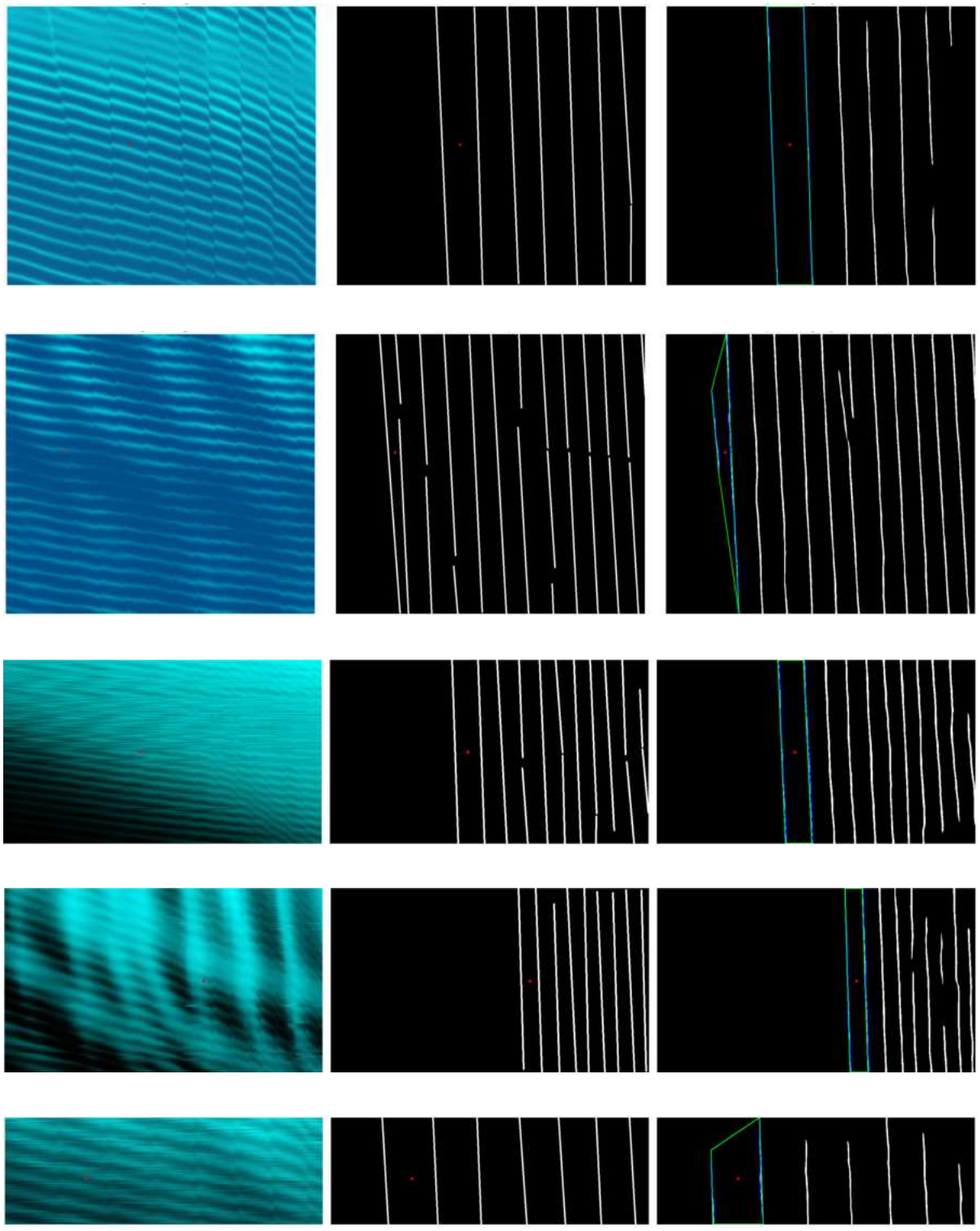


Figure B.2: Additional successful inference examples (part B).

Bibliography

[1] Max Planck. "Zur theorie des gesetzes der energieverteilung im normalspektrum". In: *Berlin* (1900), pp. 237–245.

[2] Albert Einstein. "Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt". In: *Annalen der Physik* 322.6 (1905), pp. 132–148.

[3] Niels Bohr. "On the Constitution of Atoms and Molecules". In: *Philosophical Magazine* 26.6 (July 1913), pp. 1–25.

[4] Louis De Broglie. "Recherches sur la théorie des quanta". PhD thesis. Migration-université en cours d'affectation, 1924.

[5] W Heisenberg. "On the quantum-theoretical reinterpretation of kinematical and mechanical relationships". In: *Z. Physik* 33 (1925), pp. 879–893.

[6] E. Schrödinger. "An Undulatory Theory of the Mechanics of Atoms and Molecules". In: *Phys. Rev.* 28 (6 Dec. 1926), pp. 1049–1070. DOI: 10.1103/PhysRev.28.1049.

[7] Werner Heisenberg. "Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik". In: *Zeitschrift für Physik* 43 (1927), pp. 172–198. DOI: 10.1007/BF01397280.

[8] Jonathan P. Dowling and Gerard J. Milburn. "Quantum technology: the second quantum revolution". In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 361.1809 (Aug. 15, 2003). Ed. by A. G. J. MacFarlane, pp. 1655–1674. ISSN: 1364-503X, 1471-2962. DOI: 10.1098/rsta.2003.1227.

[9] Gordon E Moore et al. "Progress in digital integrated electronics". In: *Electron devices meeting*. Vol. 21. Washington, DC. 1975, pp. 11–13.

[10] John Preskill. *Quantum computing and the entanglement frontier*. 2012. arXiv: 1203.5813 [quant-ph].

[11] Richard P. Feynman. "Simulating Physics with Computers". In: *International Journal of Theoretical Physics* 21.6-7 (1982), pp. 467–488. DOI: 10.1007/BF02650179.

[12] Alán Aspuru-Guzik et al. "Simulated Quantum Computation of Molecular Energies". In: *Science* 309.5741 (2005), pp. 1704–1707. DOI: 10.1126/science.1113479. eprint: https://www.science.org/doi/pdf/10.1126/science.1113479.

[13] Iulia M Georgescu, Sahel Ashhab, and Franco Nori. "Quantum simulation". In: *Reviews of Modern Physics* 86.1 (2014), pp. 153–185.

[14] Yudong Cao et al. "Quantum chemistry in the age of quantum computing". In: *Chemical reviews* 119.19 (2019), pp. 10856–10915.

[15] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[16] Vaishali Bhatia and K.R. Ramkumar. "An Efficient Quantum Computing technique for cracking RSA using Shor's Algorithm". In: *2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*. 2020, pp. 89–94. DOI: 10.1109/ICCCA49541.2020.9250806.

[17] Charles H Bennett and Gilles Brassard. "An update on quantum cryptography". In: *Workshop on the theory and application of cryptographic techniques*. Springer. 1984, pp. 475–480.

[18] Artur K. Ekert. "Quantum cryptography based on Bell's theorem". In: *Phys. Rev. Lett.* 67 (6 Aug. 1991), pp. 661–663. DOI: 10.1103/PhysRevLett.67.661.

[19] Lov K. Grover. "Quantum Mechanics Helps in Searching for a Needle in a Haystack". In: *Physical Review Letters* 79.2 (July 14, 1997), pp. 325–328. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.79.325.

[20] Peter W. Shor. "Scheme for reducing decoherence in quantum computer memory". In: *Phys. Rev. A* 52 (4 Oct. 1995), R2493–R2496. DOI: 10.1103/PhysRevA.52.R2493.

[21] Austin G Fowler et al. "Surface codes: Towards practical large-scale quantum computation". In: *Physical Review A—Atomic, Molecular, and Optical Physics* 86.3 (2012), p. 032324.

[22] John Preskill. "Quantum computing in the NISQ era and beyond". In: *Quantum* 2 (2018), p. 79.

[23] Rodney Van Meter and Dominic Horsman. "A blueprint for building a quantum computer". In: *Communications of the ACM* 56.10 (2013), pp. 84–93.

[24] Morten Kjaergaard et al. "Superconducting qubits: Current state of play". In: *Annual Review of Condensed Matter Physics* 11.1 (2020), pp. 369–395.

[25] Si-Hui Tan and Peter P. Rohde. "The resurgence of the linear optics quantum interferometer — recent advances & applications". In: *Reviews in Physics* 4 (2019), p. 100030. ISSN: 2405-4283. DOI: https://doi.org/10.1016/j.revip.2019.100030.

[26] Colin D Bruzewicz et al. "Trapped-ion quantum computing: Progress and challenges". In: *Applied physics reviews* 6.2 (2019).

[27] Daniel Loss and David P. DiVincenzo. "Quantum computation with quantum dots". In: *Phys. Rev. A* 57 (1 Jan. 1998), pp. 120–126. DOI: 10.1103/PhysRevA.57.120.

[28] R. Hanson et al. "Spins in few-electron quantum dots". In: *Rev. Mod. Phys.* 79 (4 Oct. 2007), pp. 1217–1265. DOI: 10.1103/RevModPhys.79.1217.

[29] Amina Sadik. "Intégration de matrices de boîtes quantiques sur silicium". PhD thesis. 2025.

[30] Daniel Loss and David P. DiVincenzo. "Quantum computation with quantum dots". In: *Physical Review A* 57.1 (Jan. 1, 1998). Number: 1, pp. 120–126. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.57.120.

[31] M. Veldhorst et al. "A two-qubit logic gate in silicon". In: *Nature* 526.7573 (Oct. 2015). Number: 7573, pp. 410–414. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature15263.

[32] T. F. Watson et al. "A programmable two-qubit quantum processor in silicon". In: *Nature* 555.7698 (Mar. 2018). Number: 7698, pp. 633–637. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature25766.

[33] Guido Burkard et al. "Semiconductor spin qubits". In: *Reviews of Modern Physics* 95.2 (June 14, 2023). Number: 2, p. 025003. ISSN: 0034-6861, 1539-0756. DOI: 10.1103/RevModPhys.95.025003.

[34] R. Maurand et al. "A CMOS silicon spin qubit". In: *Nature Communications* 7.1 (Nov. 24, 2016), p. 13575. ISSN: 2041-1723. DOI: 10.1038/ncomms13575.

[35] M. F. Gonzalez-Zalba et al. "Scaling silicon-based quantum computing using CMOS technology". In: *Nature Electronics* 4.12 (Dec. 20, 2021), pp. 872–884. ISSN: 2520-1131. DOI: 10.1038/s41928-021-00681-y.

[36] N. I. Dumoulin Stuyck et al. "Uniform Spin Qubit Devices with Tunable Coupling in an All-Silicon 300 mm Integrated Process". In: *2021 Symposium on VLSI Circuits*. 2021 Symposium on VLSI Circuits. Kyoto, Japan: IEEE, June 13, 2021, pp. 1–2. ISBN: 978-4-86348-780-2. DOI: 10.23919/VLSICircuits52068.2021.9492427.

[37] A. Elsayed et al. "Low charge noise quantum dots with industrial CMOS manufacturing". In: *npj Quantum Information* 10.1 (July 19, 2024), p. 70. ISSN: 2056-6387. DOI: 10.1038/s41534-024-00864-3.

[38] Juha T. Muhonen et al. "Storing quantum information for 30 seconds in a nanoelectronic device". In: *Nature Nanotechnology* 9.12 (Dec. 2014), pp. 986–991. ISSN: 1748-3387, 1748-3395. DOI: 10.1038/nnano.2014.211.

[39] Alexei M. Tyryshkin et al. "Electron spin coherence exceeding seconds in high-purity silicon". In: *Nature Materials* 11.2 (Feb. 2012), pp. 143–147. ISSN: 1476-1122, 1476-4660. DOI: 10.1038/nmat3182.

[40] M. Veldhorst et al. "An addressable quantum dot qubit with fault-tolerant control-fidelity". In: *Nature Nanotechnology* 9.12 (Dec. 2014), pp. 981–985. ISSN: 1748-3387, 1748-3395. DOI: 10.1038/nnano.2014.216.

[41] Jonathan Y. Huang et al. "High-fidelity spin qubit operation and algorithmic initialization above 1 K". In: *Nature* 627.8005 (Mar. 2024), pp. 772–777. ISSN: 1476-4687. DOI: 10.1038/s41586-024-07160-2.

[42] Luca Petit et al. "Design and integration of single-qubit rotations and two-qubit gates in silicon above one Kelvin". In: *Communications Materials* 3.1 (Nov. 2, 2022), p. 82. ISSN: 2662-4443. DOI: 10.1038/s43246-022-00304-9.

[43] Kenta Takeda et al. "A fault-tolerant addressable spin qubit in a natural silicon quantum dot". In: *Science Advances* 2.8 (Aug. 5, 2016), e1600694. ISSN: 2375-2548. DOI: 10.1126/sciadv.1600694.

[44] Jun Yoneda et al. "A quantum-dot spin qubit with coherence limited by charge noise and fidelity higher than 99.9%". In: *Nature Nanotechnology* 13.2 (Feb. 2018), pp. 102–106. ISSN: 1748-3387, 1748-3395. DOI: 10.1038/s41565-017-0014-x.

[45] Akito Noiri et al. "A shuttling-based two-qubit logic gate for linking distant silicon quantum processors". In: *Nature Communications* 13.1 (Sept. 30, 2022), p. 5740. ISSN: 2041-1723. DOI: 10.1038/s41467-022-33453-z.

[46] Xiao Xue et al. "Quantum logic with spin qubits crossing the surface code threshold". In: *Nature* 601.7893 (Jan. 20, 2022), pp. 343–347. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-021-04273-w.

[47] Adam R. Mills et al. "Two-qubit silicon quantum processor with operation fidelity exceeding 99%". In: *Science Advances* 8.14 (Apr. 8, 2022), eabn5130. ISSN: 2375-2548. DOI: 10.1126/sciadv.abn5130.

[48] David P DiVincenzo. "The physical implementation of quantum computation". In: *Fortschritte der Physik: Progress of Physics* 48.9-11 (2000), pp. 771–783.

[49] T. Edvinsson. "Optical quantum confinement and photocatalytic properties in two-, one- and zero-dimensional nanostructures". In: *Royal Society Open Science* 5.9 (Sept. 2018), p. 180387. ISSN: 2054-5703. DOI: 10.1098/rsos.180387.

[50] David L. Klein et al. "An approach to electrical studies of single nanocrystals". In: *Applied Physics Letters* 68.19 (1996), pp. 2574–2576. DOI: 10.1063/1.116188.

[51] M. Urdampilleta et al. "Supramolecular spin valves". In: *Nature Materials* 10 (2011), pp. 502–506. DOI: 10.1038/nmat3050.

[52] Cees Dekker. "Carbon Nanotubes as Molecular Quantum Wires". In: *Physics Today* 52.5 (1999), pp. 22–28. DOI: 10.1063/1.882658.

[53] Leo P. Kouwenhoven, D. G. Austing, and Seigo Tarucha. "Few-electron quantum dots". In: *Reports on Progress in Physics* 64.6 (2001), pp. 701–736. DOI: 10.1088/0034-4885/64/6/201.

[54] Leo P. Kouwenhoven et al. "Electron Transport in Quantum Dots". In: *Mesoscopic Electron Transport*. Ed. by L. L. Sohn, L. P. Kouwenhoven, and G. Schön. Kluwer, 1997, pp. 105–214. DOI: 10.1007/978-94-015-8839-3_4.

[55] Floris A. Zwanenburg et al. "Silicon quantum electronics". In: *Rev. Mod. Phys.* 85 (3 July 2013), pp. 961–1019. DOI: 10.1103/RevModPhys.85.961.

[56] Emmanuel Chanrion. "Charge control in semiconductor quantum-dot arrays and prospects for large-scale integration". PhD thesis. Université Grenoble Alpes [2020-....], 2021.

[57] Brett M Maune et al. "Coherent singlet-triplet oscillations in a silicon-based double quantum dot". In: *Nature* 481.7381 (2012), pp. 344–347.

[58] Maximilian Russ and Guido Burkard. "Three-electron spin qubits". In: *Journal of Physics: Condensed Matter* 29.39 (2017), p. 393001.

[59] Dohun Kim et al. "Quantum control and process tomography of a semiconductor quantum dot hybrid qubit". In: *Nature* 511.7507 (2014), pp. 70–74.

[60] B. J. van Wees et al. "Quantized conductance of point contacts in a two-dimensional electron gas". In: *Phys. Rev. Lett.* 60 (9 Feb. 1988), pp. 848–850. DOI: 10.1103/PhysRevLett.60.848.

[61] M. Field et al. "Measurements of Coulomb blockade with a noninvasive voltage probe". In: *Phys. Rev. Lett.* 70 (9 Mar. 1993), pp. 1311–1314. DOI: 10.1103/PhysRevLett.70.1311.

[62] M. A. Kastner. "The single-electron transistor". In: *Rev. Mod. Phys.* 64 (3 July 1992), pp. 849–858. DOI: 10.1103/RevModPhys.64.849.

[63] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[64] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep feedforward networks". In: *Deep learning* 1 (2016), pp. 161–217.

[65] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[66] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[67] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[68] Pedro L. Fernández-Cabán, Forrest J. Masters, and Brian M. Phillips. "Predicting Roof Pressures on a Low-Rise Structure From Freestream Turbulence Using Artificial Neural Networks". In: *Frontiers in Built Environment* Volume 4 - 2018 (2018). ISSN: 2297-3362. DOI: 10.3389/fbuil.2018.00068.

[69] Sunanda Das et al. "A Voting Approach for Heart Sounds Classification Using Discrete Wavelet Transform and CNN Architecture". In: *SN Computer Science* 5.2 (Feb. 5, 2024), p. 251. ISSN: 2661-8907. DOI: 10.1007/s42979-023-02580-9.

[70] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1, 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[71] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[72] Anders Krogh and John Hertz. "A simple weight decay can improve generalization". In: *Advances in neural information processing systems* 4 (1991).

[73] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 1, 2014), pp. 1929–1958. ISSN: 1532-4435.

[74] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (July 6, 2019), p. 60. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0.

[75] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[76] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 27. 2014, pp. 3320–3328.

[77] Walber. *Precision and recall*. https://commons.wikimedia.org/wiki/File:Precisionrecall.svg. Own work; licensed under Creative Commons Attribution–ShareAlike 4.0 International (CCBY-SA4.0). Nov. 2014.

[78] Justyna P. Zwolak and Jacob M. Taylor. "*Colloquium* : Advances in automation of quantum dot devices control". In: *Reviews of Modern Physics* 95.1 (Feb. 17, 2023), p. 011006. ISSN: 0034-6861, 1539-0756. DOI: 10.1103/RevModPhys.95.011006.

[79] Joshua Ziegler et al. "Tuning Arrays with Rays: Physics-Informed Tuning of Quantum Dot Charge States". In: *Physical Review Applied* 20.3 (Sept. 28, 2023), p. 034067. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.20.034067.

[80] Joshua Ziegler et al. "Toward Robust Autotuning of Noisy Quantum dot Devices". In: *Physical Review Applied* 17.2 (Feb. 25, 2022), p. 024069. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.17.024069.

[81] Hanwei Liu et al. "An automated approach for consecutive tuning of quantum dot arrays". In: *Applied Physics Letters* 121.8 (Aug. 22, 2022), p. 084002. ISSN: 0003-6951, 1077-3118. DOI: 10.1063/5.0111128.

[82] H. Moon et al. "Machine learning enables completely automatic tuning of a quantum device faster than human experts". In: *Nature Communications* 11.1 (Aug. 19, 2020), p. 4161. ISSN: 2041-1723. DOI: 10.1038/s41467-020-17835-9.

[83] Justyna P. Zwolak et al. "Autotuning of Double-Dot Devices *In Situ* with Machine Learning". In: *Physical Review Applied* 13.3 (Mar. 31, 2020), p. 034075. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.13.034075.

[84] B. Severin et al. "Cross-architecture tuning of silicon and SiGe-based quantum devices using machine learning". In: *Scientific Reports* 14.1 (July 27, 2024), p. 17281. ISSN: 2045-2322. DOI: 10.1038/s41598-024-67787-z.

[85] Sandesh S. Kalantre et al. "Machine learning techniques for state recognition and auto-tuning in quantum dots". In: *npj Quantum Information* 5.1 (Jan. 21, 2019), p. 6. ISSN: 2056-6387. DOI: 10.1038/s41534-018-0118-7.

[86] J Darulová, M Troyer, and M C Cassidy. "Evaluation of synthetic and experimental training data in supervised machine learning applied to charge-state detection of quantum dots". In: *Machine Learning: Science and Technology* 2.4 (Dec. 1, 2021), p. 045023. ISSN: 2632-2153. DOI: 10.1088/2632-2153/ac104c.

[87] Justin K Perron, M D Stewart Jr, and Neil M Zimmerman. "A quantitative study of bias triangles presented in chemical potential space". In: *Journal of Physics: Condensed Matter* 27.23 (June 17, 2015), p. 235302. ISSN: 0953-8984, 1361-648X. DOI: 10.1088/0953-8984/27/23/235302.

[88] T. Hensgens. "Emulating Fermi-Hubbard physics with quantum dots". PhD thesis. Delft University of Technology, 2018. DOI: 10.4233/UUID:B71F3B0B-73A0-4996-896C-84ED43E72035.

[89] T. A. Baart et al. "Computer-automated tuning of semiconductor double quantum dots into the single-electron regime". In: *Applied Physics Letters* 108.21 (May 24, 2016), p. 213104. ISSN: 0003-6951. DOI: 10.1063/1.4952624.

[90] M. Lapointe-Major et al. "Algorithm for automated tuning of a quantum dot into the single-electron regime". In: *Physical Review B* 102.8 (Aug. 3, 2020). Publisher: American Physical Society, p. 085301. DOI: 10.1103/PhysRevB.102.085301.

[91] R. Durrer et al. "Automated Tuning of Double Quantum Dots into Specific Charge States Using Neural Networks". In: *Physical Review Applied* 13.5 (May 8, 2020), p. 054019. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.13.054019.

[92] Stefanie Czischek et al. "Miniaturizing neural networks for charge state autotuning in quantum dots". In: *Machine Learning: Science and Technology* 3.1 (Mar. 1, 2022), p. 015001. ISSN: 2632-2153. DOI: 10.1088/2632-2153/ac34db.

[93] Victor Yon et al. "Robust quantum dots charge autotuning using neural network uncertainty". In: *Machine Learning: Science and Technology* 5.4 (Dec. 1, 2024), p. 045034. ISSN: 2632-2153. DOI: 10.1088/2632-2153/ad88d5.

[94] Victor Yon et al. "Experimental Online Quantum Dots Charge Autotuning Using Neural Networks". In: *Nano Letters* 25.10 (Mar. 12, 2025). Publisher: American Chemical Society, pp. 3717–3725. ISSN: 1530-6984. DOI: 10.1021/acs.nanolett.4c04889.

[95] Jonas Schuff et al. *Fully autonomous tuning of a spin qubit*. Version Number: 1. 2024. DOI: 10.48550/ARXIV.2402.03931.

[96] Julian D. Teske et al. "A machine learning approach for automated fine-tuning of semiconductor spin qubits". In: *Applied Physics Letters* 114.13 (Apr. 1, 2019), p. 133102. ISSN: 0003-6951, 1077-3118. DOI: 10.1063/1.5088412.

[97] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28.

[98] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 4510–4520.

[99] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[100] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. 4th. Pearson, 2018.

Author Declarations

Statement of Non-Plagiarism

I hereby declare that all information in this report has been obtained and presented in accordance with academic rules and ethical conduct. The work I am submitting in this report, except where otherwise indicated, is entirely my own.

AI Usage Statement

Generative AI tools such as large language models and AI coding assistants were used to support the preparation of this thesis. These tools assisted with drafting initial sections of the manuscript, generating parts of the software stack, and its documentation. All AI-generated text and code were critically reviewed, edited, tested, and validated by the author. The author retains full responsibility for the accuracy, interpretation, and integrity of the final content.

Copyright and Permissions

All figures, tables, and illustrations included in this report that are not of my own creation have been used with the appropriate permissions from the original copyright holders. No copyrighted material has been used without proper authorization.

Date: August 22, 2025

Signature: 

Supervisors Approval

I, the undersigned, **Yann Beilliard**, supervisor of **Peter Samaha**, student of the PSRS EMJMD, during his master thesis at CEA-Leti, certify that I approve the content of this master thesis report entitled: *"Neural Network Segmentation of Charge Stability Diagrams for the Auto-Tuning of Silicon Quantum Dots for Spin Qubits."*

Date: 22/08/2025

Signature: 

I, the undersigned, **Pierre-André Mortemousque**, supervisor of **Peter Samaha**, student of the PSRS EMJMD, during his master thesis at CEA-Leti, certify that I approve the content of this master thesis report entitled: *'Neural Network Segmentation of Charge Stability Diagrams for the Auto-Tuning of Silicon Quantum Dots for Spin Qubits."*

Date: August 22, 2025

Signature: