Bridging Simulation and Experiment in Nanoscience with AI

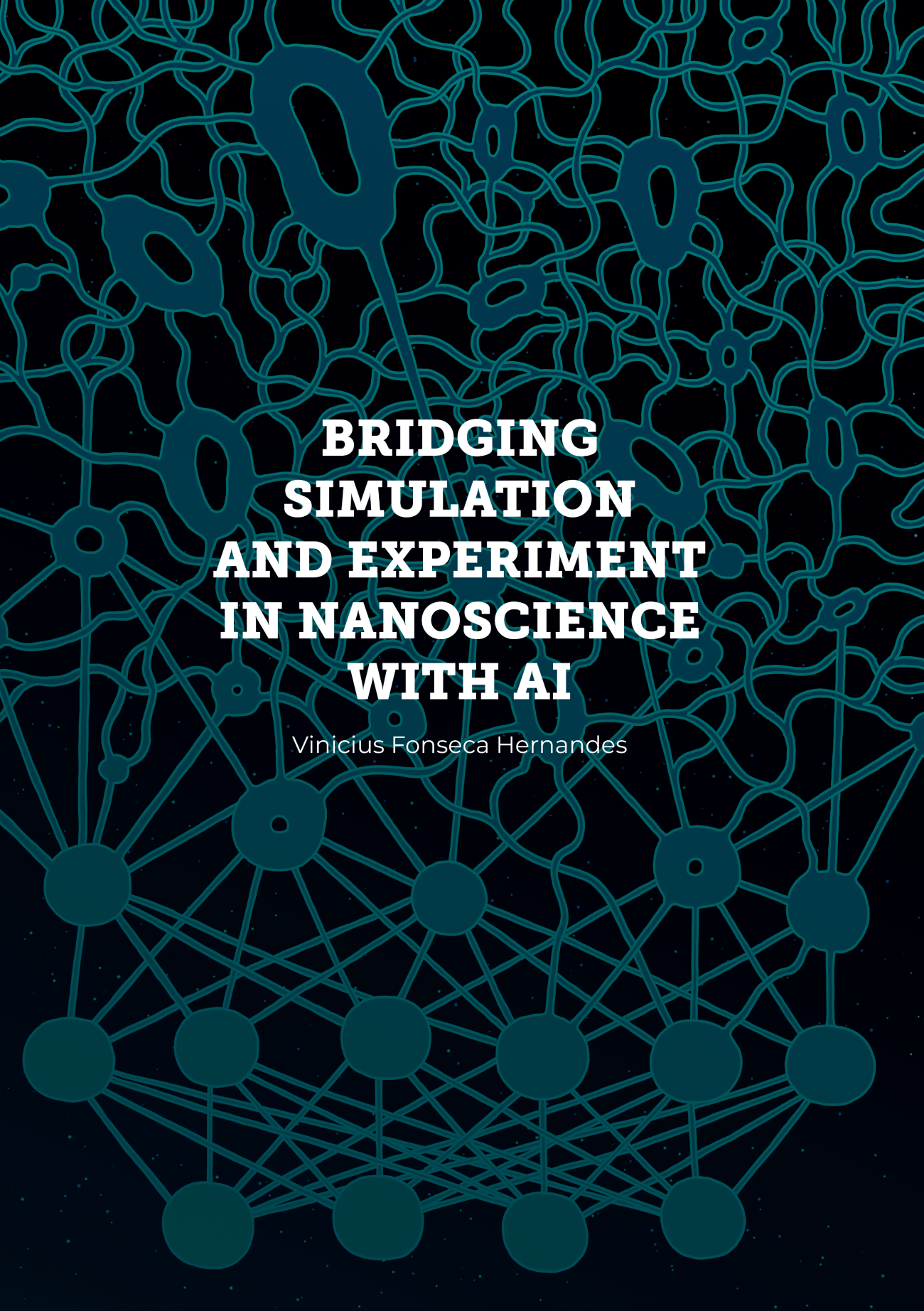
Fonseca Hernandes, V.

DOI
10.4233/uuid:4be6e7b1-6a79-4136-9d3c-a46c582c4752

Publication date
2025

Document Version
Final published version

Citation (APA)
Fonseca Hernandes, V. (2025). *Bridging Simulation and Experiment in Nanoscience with AI*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:4be6e7b1-6a79-4136-9d3c-a46c582c4752



BRIDGING SIMULATION AND EXPERIMENT IN NANOSCIENCE WITH AI

Vinicius Fonseca Hernandes

BRIDGING SIMULATION AND EXPERIMENT IN NANOSCIENCE WITH AI

BRIDGING SIMULATION AND EXPERIMENT IN NANOSCIENCE WITH AI

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board of Doctorates,
to be defended publicly on
Friday 19 of December 2025 at 10:00 o'clock

by

Vinicius FONSECA HERNANDES

Master of Science in Physics,
Federal University of Pelotas, Brazil,
born in Pelotas, Brazil.

This dissertation has been approved by the promoters and supervisors.

promotor: Prof. dr. ir R. Hanson
copromotor: Dr. E. Greplová

Composition of the doctoral committee:

Rector Magnificus	chairperson
Prof. dr. ir. R. Hanson	Delft University of Technology
Dr. E. Greplová	Delft University of Technology
Independent members:	
Prof. dr. J. Carrasquilla Alvarez	ETH Zurich, Switzerland
Dr. A. M. Dawid	Leiden University
Prof. dr. ir. L. P. Kouwenhoven	Delft University of Technology
Dr. ir. M. Veldhorst	Delft University of Technology
Prof. dr. ir. H. S. J. van der Zant	Delft University of Technology, *reserve member*

Keywords: Machine Learning, Neural Networks, Quantum Generative Learning, Condensed Matter, Neuroscience, Spin Qubits, Quantum Dots

Printed by: Gildeprint

Front & Back: Design by Arthur Feltraco

ISBN 978-94-6496-492-9

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

Knowledge emerges only through invention and re-invention, through the restless, impatient, continuing, hopeful inquiry human beings pursue in the world, with the world, and with each other.

Paulo Freire

Contents

SUMMARY

This thesis explores how artificial intelligence (AI) can be used to bridge the gap between simulation and experiment in nanoscience. As both theoretical modeling and experimental techniques in nanoscience become increasingly sophisticated, AI is emerging as a powerful tool to tackle challenges such as tuning experiments, accelerating simulations, generating synthetic data, and automating data analysis. This work presents applications of AI in three main domains: neuroscience, quantum computing, and condensed matter physics.

In neuroscience, we address the problem of efficiently simulating neuronal activity data by implementing a quantum machine learning model that uses a reduced number of trainable parameters. On the experimental side, we develop a computational package that automates the analysis of micro-electrode array data using a neural network trained to replicate human expert detection of burst patterns from spiking activity.

In spin-based quantum computing, we develop a computational package that simulates charge stability diagrams (CSDs) based on device characteristics, enabling the efficient creation of synthetic datasets. We also demonstrate how machine learning models can be used to filter high-quality CSDs for building experimental datasets, and we present the first implementation of diffusion models to complete partially measured CSDs, an approach that can be integrated into measurement routines to accelerate the process.

In condensed matter physics, we leverage neural quantum states to detect phase transitions by analyzing the evolution of neural network weights, without the need to calculate order parameters, and discuss potential directions for combining this technique with neural quantum states trained on experimental data.

These studies show that AI can accelerate simulations and data analysis, while also supporting the design and interpretation of experiments, highlighting its growing and essential role in the future of nanoscience.

SAMENVATTING

Deze thesis onderzoekt hoe kunstmatige intelligentie (AI) kan worden ingezet om de kloof tussen simulatie en experiment in de nanowetenschap te overbruggen. Naarmate zowel theoretische modellering als experimentele technieken in de nanowetenschap steeds geavanceerder worden, ontwikkelt AI zich tot een krachtig hulpmiddel om uitdagingen aan te pakken zoals het afstellen van experimenten, het versnellen van simulaties, het genereren van synthetische data en het automatiseren van data-analyse. Dit werk presenteert toepassingen van AI in drie hoofdgebieden: neurowetenschappen, kwantumcomputing en gecondenseerde-materie-fysica.

In de neurowetenschappen richten we ons op het efficiënt simuleren van neuronale activiteitsdata door een kwantum-machine learningmodel te implementeren met een gereduceerd aantal trainbare parameters. Aan de experimentele kant ontwikkelen we een computationeel pakket dat de analyse van micro-elektrode-arraydata automatiseert met behulp van een neuraal netwerk dat getraind is om de detectie van burstpatronen uit spikingactiviteit door menselijke experts na te bootsen.

In spin-gebaseerde kwantumcomputing ontwikkelen we een computationeel pakket dat ladingsstabiliteitsdiagrammen (CSDs) simuleert op basis van apparaatkenmerken, waardoor het efficiënt genereren van synthetische datasets mogelijk wordt. We demonstreren ook hoe machine learning-modellen gebruikt kunnen worden om CSDs van hoge kwaliteit te filteren voor het samenstellen van experimentele datasets, en presenteren de eerste implementatie van diffusiemodellen om gedeeltelijk gemeten CSDs aan te vullen, een aanpak die geïntegreerd kan worden in meetroutines om het proces te versnellen.

In de gecondenseerde-materie-fysica benutten we neurale kwantumtoestanden om faseovergangen te detecteren door de evolutie van neurale netwerkgewichten te analyseren, zonder dat berekening van ordeparameters nodig is. We bespreken tevens mogelijke richtingen om deze techniek te combineren met op experimentele data getrainde neurale kwantumtoestanden.

Deze studies tonen aan dat AI simulaties en data-analyse kan versnellen, en tegelijkertijd kan bijdragen aan het ontwerp en de interpretatie van experimenten, wat de groeiende en essentiële rol van AI in de toekomst van de nanowetenschap onderstreept.

1

INTRODUCTION

Nanoscience refers to the study, manipulation, and engineering of matter at the nanometer scale. Several topics fall into this definition, from the control of individual atoms to be used as advances sensor or for quantum computation, to the understanding of molecular mechanisms that lead to cellular functions. In the last decades, several outstanding advances in nanoscience occurred. In the field of quantum science and technology, some examples include the universal control of semiconductor based quantum processors [1], long distance entanglement color centers in diamond [2], and efficient control of spin-waves [3]. In bionanoscience, notable advancements include steps towards the formation of synthetic cells [4], and the correct establishment of synaptic connections in order to understand neural circuitry formation [5].

1.1. SIMULATION AS A TECHNIQUE TO SOLVE SCIENTIFIC PROBLEMS

One tool that sped up significantly the pace of scientific advances in nanoscience - and in any other scientific discipline - is the computer. Since their advent computers are being heaviliy used for solving problems in nanoscience. Historically, the use of computers has first been applied in physics in the 1940s and 1950s to simulate complex atomic phenomena using the Monte Carlo method, developed by Metropolis and Ulam [6]. This probabilistic approach allowed physicists to study processes such as neutron transport, that was previously inaccessible by analytical methods [7]. From the 1960s to the 1980s the world witnessed the rise of computational chemistry and material science. Molecular dynamics simulations allowed the atomistic modelling of materials and fluids [8, 9]. In 1985, the Car and Parrinello method [10] unified molecular dynamics with density functional theory, allowing the simulation of quantum mechanical systems with high precision. This had a profound impact in nanoscience, enabling prediction of molecular structures and nanoscale interactions.

At the same time, neuroscience started implementing computational methods to simulate brain activity. The Hodgkin-Huxley model for a neuron, originally proposed

as a mathematical framework in 1952 [11], became widely simulated using some of the most powerful digital computers available at the time [12, 13]. In the 1980s and 1990s, computational neuroscience solidified itself as a field, focusing on topics such as neural plasticity, neural networks, and spiking dynamics [14, 15, 16].

In parallel, the field of quantum computing was born. From 1980 to 1982, Paul Benioff and Richard Feynman proposed the use of quantum machine to simulate quantum matter [17] and perform general computation [18, 19]. This later led to the formulations on algorithms designed to run on quantum circuits, like the Shor's algorithm for factorization [20], Grover's algorithm for search [21], and Kitaev's method for phase estimation [22]. The latter was particularly important for the development of the field now known as quantum simulation - using quantum systems to emulate complex many-body Hamiltonians - opening a new path to the simulation of nanoscale physical systems and materials [23].

1.2. EXPERIMENTS IN NANOSCIENCE: FROM QUANTUM COMPUTING TO NEUROSCIENCE

The significant advances in theory and simulation in nanoscience were accompanied by several field-changing experimental discoveries, which throughout the 20th century evolved from analysis of bulk properties of materials to control and manipulation of individual atoms. Early works at the beginning of the 20th century started using X-ray diffraction to infer structural properties of materials [24], and electron microscopes to visualize materials' surfaces and morphology [25]. Later, the discovery of the Scanning Tunneling Microscope (STM) allowed scientists to produce images of surfaces with subnanometer resolution [26]. In 1990 the same method was used to move atoms around a surface, allowing engineering of atomic patterns [27]. Advances in lithography, first in the 1970s with electron-beam lithography and later with extreme ultraviolet lithography, enabled the fabrication of devices with sizes below 10 nm [28, 29].

The realization of computation using quantum bits (qubits) also developed significantly during the last decades. Qubits can be implemented using different physical systems, which give rise to different ways on how to construct a quantum computer. After the Cirac-Zoller proposed scheme for quantum computing [30], in 1995, the first demonstration of a quantum logic gate is proposed using a single trapped ion [31], and later expanded to two and four ions [32, 33]. Quantum computation using nuclear magnetic resonance was first demonstrated in 1997 [34, 35], followed in the next years by the first experimental realization of Grover [36]and Shor [37] algorithms. Superconductor based quantum computers were also proposed around the same time [38], with rapid advances in the beginning of the 21st century [39, 40]; a similar timeline was followed by the use of quantum dots as qubits, with its first proposal in 1998 by Loss and di Vincenzo [41], and significant progress in the early 2000s [42, 43, 44]. Each one of these quantum computing platforms offer different advantages and face distinct implementation challenges, and since their inception, they have been advancing in parallel, sometimes competing, and often collaborating. Despite their differences, all of the platforms showed outstanding progress. In 2025 we have superconducting based quantum computers reaching one hundred and more qubits, and with error rates below the surface code threshold [45,

46, 47]; and spin quantum dot based chips achieving high-fidelity gates [48], universal control [1], capable at operating at high temperatures [49, 50], and fabrication processes that leverage compatibility with industrial settings [51].

Another sub-field of nanoscience, and specifically bionanoscience, that was transformed by novel experimental techniques in the last 50 years, is neurophysiology. One of the main experimental challenges in neuroscience in recording the activity of individual neurons. The first steps towards this goal started in the 1950 with microelectrodes being used to recording electrical signals from the cerebral cortex [52, 53, 54]. In the late 70s, the patch-clamp technique by Neher and Sakmann which made possible the recording of electrical currents in ion channel molecules [55]. Concurrently, the first implementation of Micro-Electrode Arrays (MEAs) to measure electrical activity in cultured cells took place [56, 57]. Unlike electroencephalography (EEG), which records global brain activity from the scalp and, while giving informative data about the functional brain organization and connectivity, offers only coarse spatial resolution, the micro-electrode-based techniques can capture the activity of individual neurons or small circuits with millisecond precision [58, 59, 60]. In the last decade, experimental breakthroughs in nanotechnology enhanced the capability of tools used to record neuronal activity with increasing precision [61]. Some examples of this integration between nanoscience and neuroscience include the development of nanomaterials-based MEAs [62, 63] and the use of nitrogen-vacancy centers as sensors for nanometer scale measurement of neuronal activity [64, 65].

1.3. SIMULATION - EXPERIMENT SYNERGY

The experimental and theoretical advances cited show how the field of nanoscience came to its current state, however, we did not specify how these two approaches work in synergy: experiments require interpretations and insights given by theoretical models, and simulations need experimental validation. The development of nanomaterials is a perfect example of this collaboration: when designing a new material with specific desired properties, a first step often consists of analyzing theoretical predictions, using methods such as density functional theory [66, 67] to calculate the material electronic band structure and its Raman spectrum, for example, to guide experimentalists towards the most promising synthesis routes. After synthetizing the material, experimental characterization using techniques like X-ray absorption spectroscopy [68] and Raman spectroscopy [69] can be used to confirm theoretical predictions, or reveal discrepancies between the simulation and experiment, which can then be used to either refine the theoretical modeling or the synthesis strategy. In condensed matter physics, the case of topological insulators [70] are one exceptional demonstration of the theory-experiment collaboration, with theoretical studies predicting the existing of such materials [71, 72, 73], followed by the experimental observations confirming the theoretical findings [74, 75]. The field of quantum computing is full of examples where a theory-experiment feedback loop takes place, from which we chose two cases to discuss in this thesis. One of them is error correction, crucial to build fault-tolerant quantum computers. Error correcting codes are first proposed as mathematical frameworks, like the toric and the surface code [76, 77, 78], and consecutively tested in experimental implementations [79, 80], which provide feedback the new theoretical models, like incorporating better noise models of

the devices when running the error correcting scheme [81, 82]. Similarly, the design of novel qubit architectures often consists of theoretical proposals, like for the superconducting transmons qubits and the silicon spin qubit [83, 41], followed by experimental fabrication and characterization of the design [84, 85], from which noise sources and coherence times and be calculated and used to improve qubits with improved design and material composition that mitigate the experimental challenges [86, 87]. In neuroscience, the synergy between computational models and experiments is also crucial to decode the complex functions of the brain at the nano scale. In the case of synaptic plasticity, for example, computational models of neural networks are used to propose rules of how synaptic strength change [88, 89]. Afterwards, data coming from measurements of neuronal activity [90, 91] is used to developed refine theoretical models [92, 93].

1.4. AI FOR NANOSCIENCE

Both the theoretical and experimental methods applied to solve scientific problems in nanoscience were greatly impacted by a development that has become an everyday agent in modern life, the use of artificial intelligence (AI) models. AI impacted the entire scientific workflow, from the design of experiments and the analysis of results to the acceleration of computational simulations [94, 95, 96]. On the simulation side, AI enhanced the efficiency and accuracy of several techniques across different subfields of nanoscience. One seminal example is AlphaFold, an AI model that obtained unprecedented performance in predicting protein structure based on amino acid sequences [97], revolutionizing the field of molecular nanoscience, achievement for which the AlphaFold main proposers received the 2024 Nobel Prize in Chemistry [98]. In condensed matter physics, AI has created entire new subfield. The topic of machine learning force fields uses AI to approximate potential energy surfaces as a way to replace expensive ab initio simulations in molecular dynamics simulations, enabling the modeling of larger systems and achieving greater accuracy [99, 100, 101]. Neural Quantum States are another example of an AI application that created a thriving scientific community; AI is used to approximate the quantum wave function of physical systems, critical for ultimately understanding and developing novel quantum materials [102, 103]. The technique showed as a strong competitor to more traditional alternatives, such as Tensor Networks [104] or Quantum Monte Carlo [105, 106], and in some cases, outperforming them and obtaining state of the art simulation energies [107, 108].

In quantum computing and technology, AI gave rise of a whole new subfield again: quantum machine learning, which consists or running AI algorithms on quantum circuits. On the experimental side, AI has been used to speed-up measurements by automating the selection of important samples [109, 110], tuning routines in quantum dots devices [111, 112, 113], and extract Hamiltonian parameters from measurements of quantum simulations [114, 115, 116], between several other applications [117]. In computational neuroscience, AI models have emerged as an alternative to simulating neuronal activity with physics or biology informed models [118, 119, 120], showing great generalization and accuracy [121, 122, 123]. Concurrently, AI stands out at automating the analysis of high-dimensional neuronal data coming from experimental measurements [124]: from the reconstruction of connectomes [125, 126] to the identification and sorting of neuronal spikes [127, 128, 129].

1.5. THESIS OUTLINE

In this thesis, we use AI to bridge simulation and experiments in nanoscience, by expanding the literature on several topics in which AI models are applied to solve problems related to the simulation of neuronal and quantum data, and the automation of data analysis in neuronal and quantum experiments. We divide the thesis in three main parts, each related to a specific field of nanoscience. Each part combines a simulation-based study with an experimental or application-oriented component, except in the last part, where the focus remains theoretical and the experimental outlook is discussed in the conclusion. An overview of the thesis structure can be seen in Table 1.1, where the parts-field are linked to the correspondent chapters.

Table 1.1: Structure of the thesis, divided in three parts, each related to a specific field, with related chapters, divided in simulation and experiment-focus.

Field	**Simulation**	**Experiment**
Neuroscience	Exploring Biological Neuronal Correlations with Quantum Generative Models [Chapter 3]	autoMEA: Machine learning-based burst detection for multi-electrode array datasets [Chapter 4]
Quantum Computing and Technology	QDsim: A user-friendly toolbox for simulating large-scale quantum dot devices [Chapter 5]	Diffusion CSD [Chapter 6]
Condensed Matter Physics	Adiabatic Fine-Tuning of Neural Quantum States Enables Detection of Phase Transitions in Weight Space [Chapter 7]	Outlook [Chapter 8]

In Part 1, we show two works related to the field of neuroscience. In Chapter 3, the simulation work consists of an implementation of a quantum generative learning model to simulate neuronal activity. We show that it is possible to reduce the number of parameters necessary to simulate neuronal activity by using quantum models instead of classical deep learning alternatives. In Chapter 4, we show an application of AI to automate experimental data analysis. Specifically, we automate the process of burst detection in Micro-Electrode Array datasets, using a neural network trained on human-labeled data, capable of replace most of the human job necessary in the data analysis pipeline, and therefore drastically reducing experiment time.

Part 2 is in the field of Quantum Computing and Technology, specifically spin quantum dot devices. In Chapter 5 we introduce QDSim, a charge stability diagram simulator, tailored to simulate large arrays of quantum dots. The main goal of this package is to have a fast data generator, useful to construct large datasets useful to train machine learning models to automate experimental processes. Chapter 6 is directly related to this automation endeavor. We first design a charge stability diagram classifier, trained on a small human-labeled dataset, capable of distinguishing samples that are considered

useful or not to be used by experimentalists. We then show first results on the application of generative models to complete missing data in charge stability diagrams. This model can be exploited to speed-up measurement routines in spin quantum devices.

In Part 3 we dive into the field of Condensed Matter Physics. In Chapter 7, we show an application of weight space learning to detect phase transition in quantum systems when simulated using neural quantum states. We train neural networks to represent the ground states of a quantum Hamiltonian across a phase diagram, using a fine-tuning scheme to keep weights between different sections of the phase diagram correlated. We then analyze the mapping of the networks' weights to a lower dimensional space, where it is straightforward to find the phase transition of the system. A possible experiment application of AI in Condensed Matter is discussed in the Outlook and Conclusion chapter.

Since the thesis covers several topics that are not always directly related to each other, we give a general introduction to some of the main concepts necessary to understand the remaining of the thesis in Chapter 2. First, in Section 1, we introduce the topic of artificial intelligence and deep learning, which is central throughout the whole thesis. We briefly discuss neural networks architectures, backpropagation, and generative learning models. Then, in Section 2, we expand on the role of neural codes and spiking activity in neuroscience. We review computational methods used to simulate neuronal activity, both physics-informed methods an machine learning based methods. We finalize the section describing micro-electrode array measurements, and the challenges of analyzing this data of data. In Section 3, we give a general overview of quantum computing, focused on quantum dots based devices; we explain what are charge stability diagrams and challenges related to their measurement. In Section 4, we unify elements from Section 1 and 3 to illustrate the concept of quantum machine learning and quantum generative learning models. Ending the chapter, Section 5 introduce neural quantum states: how we can use machine learning models to approximate the wavefunction of quantum systems. We describe the restricted Boltzmann machine architecture, and outline how we can exploit ideas from neural networks interpretability and machine learning theory to learn useful information from the weights of neural quantum states.

1.6. AUTHOR'S CONTRIBUTIONS

In this section, I will clarify my contributions to the chapters of the thesis which were published as articles.

In Chapter 3, the idea of simulating neuronal activity using quantum models was conceived by Eliska Greplova. Later, we jointly explored different approaches, eventually settling on using quantum generative adversarial networks. From there, I ran all the simulations, performed the data analysis, and created the figures. The results were discussed with Eliska, who also helped with writing and revising the paper.

The idea for the project in Chapter 4 was conceived by Eliska and myself during a meeting with Dimphna Meijer, Geeske van Woerden, and other students, in which we were discussing prospects for the project in Chapter 3. All experimental data was collected by Anouk Heuvelmans. Valentina Gualtieri conducted the initial simulations and data analysis for her Master's thesis. After that, I carried out new computational simulations and designed the Python package. Anouk and I performed the data analysis and plotted all the figures. The results were discussed with all the authors, who also con-

tributed to manuscript revisions, starting from a first draft written by Anouk and myself.

The proposal to build a charge stability diagram simulator came from Eliska. Initial simulations, not included in the final paper, were run by Maia Rigot while interning at QMAI. The first version of the results and code that appear in the paper shown in Chapter 5 was developed by Charles Renshaw-Whitman during his Master's thesis in our group, co-supervised by me. After that, I took the code written by Charles and developed the first version of QDSim as a Python package. Valentina completed the project by optimizing the code, adding new functions, plotting the results, and writing the main manuscript. Eliska and I supervised the project from start to finish, contributing to discussions about the results and providing feedback on the manuscript and package organization.

The idea of using diffusion models to generate charge stability diagrams was mine, and I proposed it to Joseph Rogers, who developed the first version of the project, which became his Master's thesis. I designed new simulations by training a new model, performed the data analysis, created the figures, and wrote the first version of the manuscript, currently in preparation. Throughout the project, results and new experimental ideas were discussed with Eliska, Thomas Spriggs, and Rouven Koch.

In Chapter 7, the initial idea of finding phase transitions by analyzing the weights of trained neural quantum states came from Eliska and Thomas, and was proposed to Saqar Khaleefah. Saqar and Thomas developed the main results for a preliminary version of the project, which formed the basis of Saqar's Master's thesis. Building on that, I developed a new methodology, ran all the new simulations, created the figures, and wrote the manuscript. Discussions about the results and manuscript preparation were held with Thomas and Eliska.

1

REFERENCES

[1] Stephan GJ Philips et al. “Universal control of a six-qubit quantum processor in silicon”. In: *Nature* 609.7929 (2022), pp. 919–924.

[2] Bas Hensen et al. “Loophole-free Bell inequality violation using electron spins separated by 1.3 kilometres”. In: *Nature* 526.7575 (2015), pp. 682–686.

[3] M Borst et al. “Observation and control of hybrid spin-wave–Meissner-current transport modes”. In: *Science* 382.6669 (2023), pp. 430–434.

[4] Lorenzo Olivi et al. “Towards a synthetic cell cycle”. In: *Nature communications* 12.1 (2021), p. 4531.

[5] Dimphna H Meijer et al. “Teneurin4 dimer structures reveal a calcium-stabilized compact conformation supporting homomeric trans-interactions”. In: *The EMBO journal* 41.9 (2022), e107505.

[6] Nicholas Metropolis and Stanislaw Ulam. “The monte carlo method”. In: *Journal of the American statistical association* 44.247 (1949), pp. 335–341.

[7] W Goad and R Johnston. “A monte carlo method for criticality problems”. In: *Nuclear Science and Engineering* 5.6 (1959), pp. 371–375.

[8] JB Gibson et al. “Dynamics of radiation damage”. In: *Physical Review* 120.4 (1960), p. 1229.

[9] Aneesur Rahman. “Correlations in the motion of atoms in liquid argon”. In: *Physical review* 136.2A (1964), A405.

[10] Richard Car and Michele Parrinello. “Unified approach for molecular dynamics and density-functional theory”. In: *Physical review letters* 55.22 (1985), p. 2471.

[11] Alan L Hodgkin and Andrew F Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of physiology* 117.4 (1952), p. 500.

[12] KS Cole, HA Antosiewicz, and P Rabinowitz. “Automatic computation of nerve excitation”. In: *Journal of the Society for Industrial and Applied Mathematics* 3.3 (1955), pp. 153–172.

[13] R FitzHugh and HA Antosiewicz. “Automatic computation of nerve excitation—detailed corrections and additions”. In: *Journal of the Society for Industrial and Applied Mathematics* 7.4 (1959), pp. 447–458.

[14] James H Schwartz et al. *Principles of neural science*. Elsevier New York, 1991.

[15] Eric L Schwartz. *Computational neuroscience*. Mit Press, 1993.

[16] Frédéric Theunissen and John P Miller. “Temporal encoding in nervous systems: a rigorous definition”. In: *Journal of computational neuroscience* 2 (1995), pp. 149–162.

[17] Richard P. Feynman. “Simulating physics with computers”. In: *International Journal of Theoretical Physics* 21.6 (June 1982), pp. 467–488. ISSN: 1572-9575. DOI: 10.1007/BF02650179. URL: https://doi.org/10.1007/BF02650179.

[18] Paul Benioff. “The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines”. In: *Journal of statistical physics* 22 (1980), pp. 563–591.

[19] Paul Benioff. “Quantum mechanical Hamiltonian models of Turing machines”. In: *Journal of Statistical Physics* 29 (1982), pp. 515–546.

[20] Peter W Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[21] Lov K Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[22] A Yu Kitaev. “Quantum measurements and the Abelian stabilizer problem”. In: *arXiv preprint quant-ph/9511026* (1995).

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

[24] William Henry Bragg and William Lawrence Bragg. “The reflection of X-rays by crystals”. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 88.605 (1913), pp. 428–438.

[25] Tom Mulvey. “Origins and historical development of the electron microscope”. In: *British journal of applied physics* 13.5 (1962), p. 197.

[26] G. Binnig and H. Rohrer. “Scanning tunneling microscopy”. In: *Surface Science* 126.1 (1983), pp. 236–244. ISSN: 0039-6028. DOI: https://doi.org/10.1016/0039-6028(83)90716-1. URL: https://www.sciencedirect.com/science/article/pii/0039602883907161.

[27] Donald M Eigler and Erhard K Schweizer. “Positioning single atoms with a scanning tunnelling microscope”. In: *Nature* 344.6266 (1990), pp. 524–526.

[28] Andrew M Hawryluk and Lynn G Seppala. “Soft x-ray projection lithography using an x-ray reduction camera”. In: *Journal of Vacuum Science & Technology B: Microelectronics Processing and Phenomena* 6.6 (1988), pp. 2162–2166.

[29] Chimaobi Mbanaso and Gregory Denbeaux. “EUV Lithography”. In: *Encyclopedia of Nanotechnology*. Ed. by Bharat Bhushan. Dordrecht: Springer Netherlands, 2012, pp. 797–803. ISBN: 978-90-481-9751-4. DOI: 10.1007/978-90-481-9751-4_391. URL: https://doi.org/10.1007/978-90-481-9751-4_391.

[30] Juan I Cirac and Peter Zoller. “Quantum computations with cold trapped ions”. In: *Physical review letters* 74.20 (1995), p. 4091.

[31] Chris Monroe et al. “Demonstration of a fundamental quantum logic gate”. In: *Physical review letters* 75.25 (1995), p. 4714.

[32] QA Turchette et al. “Deterministic entanglement of two trapped ions”. In: *Physical Review Letters* 81.17 (1998), p. 3631.

[33] Cass A Sackett et al. “Experimental entanglement of four particles”. In: *Nature* 404.6775 (2000), pp. 256–259.

[34] Neil A Gershenfeld and Isaac L Chuang. “Bulk spin-resonance quantum computation”. In: *science* 275.5298 (1997), pp. 350–356.

[35] David G Cory, Amr F Fahmy, and Timothy F Havel. “Ensemble quantum computing by NMR spectroscopy”. In: *Proceedings of the National Academy of Sciences* 94.5 (1997), pp. 1634–1639.

[36] Isaac L Chuang, Neil Gershenfeld, and Mark Kubinec. “Experimental implementation of fast quantum searching”. In: *Physical review letters* 80.15 (1998), p. 3408.

[37] Lieven MK Vandersypen et al. “Experimental realization of Shor’s quantum factoring algorithm using nuclear magnetic resonance”. In: *Nature* 414.6866 (2001), pp. 883–887.

[38] Yasunobu Nakamura, Yu A Pashkin, and JS Tsai. “Coherent control of macroscopic quantum states in a single-Cooper-pair box”. In: *nature* 398.6730 (1999), pp. 786–788.

[39] Denis Vion et al. “Manipulating the quantum state of an electrical circuit”. In: *Science* 296.5569 (2002), pp. 886–889.

[40] Matthias Steffen et al. “Measurement of the entanglement of two superconducting qubits via state tomography”. In: *Science* 313.5792 (2006), pp. 1423–1425.

[41] Daniel Loss and David P DiVincenzo. “Quantum computation with quantum dots”. In: *Physical Review A* 57.1 (1998), p. 120.

[42] JM Elzerman et al. “Single-shot read-out of an individual electron spin in a quantum dot”. In: *nature* 430.6998 (2004), pp. 431–435.

[43] J. R. Petta et al. “Coherent Manipulation of Coupled Electron Spins in Semiconductor Quantum Dots”. In: *Science* 309.5744 (2005), pp. 2180–2184. DOI: 10.1126/science.1116955. eprint: https://www.science.org/doi/pdf/10.1126/science.1116955. URL: https://www.science.org/doi/abs/10.1126/science.1116955.

[44] Ronald Hanson et al. “Spins in few-electron quantum dots”. In: *Reviews of modern physics* 79.4 (2007), pp. 1217–1265.

[45] Yulin Wu et al. “Strong quantum computational advantage using a superconducting quantum processor”. In: *Physical review letters* 127.18 (2021), p. 180501.

[46] Rajeev Acharya et al. “Quantum error correction below the surface code threshold”. In: *Nature* (2024).

[47] Dongxin Gao et al. “Establishing a new benchmark in quantum computational advantage with 105-qubit Zuchongzhi 3.0 processor”. In: *Physical Review Letters* 134.9 (2025), p. 090601.

[48] Tuomo Tanttu et al. “Assessment of the errors of high-fidelity two-qubit gates in silicon quantum dots”. In: *Nature Physics* (2024), pp. 1–6.

[49] Chih Heng Yang et al. “Operation of a silicon quantum processor unit cell above one kelvin”. In: *Nature* 580.7803 (2020), pp. 350–354.

[50] Samuel Neyens et al. “Probing single electrons across 300-mm spin qubit wafers”. In: *Nature* 629.8010 (2024), pp. 80–85.

[51] AMJ Zwerver et al. “Qubits made by advanced semiconductor manufacturing”. In: *Nature Electronics* 5.3 (2022), pp. 184–190.

[52] S Woldring and DIRKEN MN. “Spontaneous unit-activity in the superficial cortical layers.” In: *Acta Physiologica et Pharmacologica Neerlandica* 1.3 (1950), pp. 369–379.

[53] Choh-Luh Li and Herbert Jasper. “Microelectrode studies of the electrical activity of the cerebral cortex in the cat”. In: *The Journal of physiology* 121.1 (1953), p. 117.

[54] JD Green. “A simple microelectrode for recording from the central nervous system”. In: *Nature* 182.4640 (1958), pp. 962–962.

[55] Erwin Neher, Bert Sakmann, and Joe Henry Steinbach. “The extracellular patch clamp: a method for resolving currents through individual open channels in biological membranes”. In: *Pflügers Archiv* 375 (1978), pp. 219–228.

[56] CA Thomas Jr et al. “A miniature microelectrode array to monitor the bioelectric activity of cultured cells”. In: *Experimental cell research* 74.1 (1972), pp. 61–66.

[57] Jerome Pine. “A history of MEA development”. In: *Advances in network electrophysiology: Using multi-electrode arrays*. Springer, 2006, pp. 3–23.

[58] Lindsay F Haas. “Hans berger (1873–1941), richard caton (1842–1926), and electroencephalography”. In: *Journal of Neurology, Neurosurgery & Psychiatry* 74.1 (2003), pp. 9–9.

[59] György Buzsáki, Costas A Anastassiou, and Christof Koch. “The origin of extracellular fields and currents—EEG, ECoG, LFP and spikes”. In: *Nature reviews neuroscience* 13.6 (2012), pp. 407–420.

[60] Marie Engelene J Obien et al. “Revealing neuronal function through microelectrode array recordings”. In: *Frontiers in neuroscience* 8 (2015), p. 423.

[61] A Paul Alivisatos et al. “Nanotools for neuroscience and brain activity mapping”. In: *ACS nano* 7.3 (2013), pp. 1850–1866.

[62] Ren Liu et al. “Ultra-Sharp Nanowire Arrays Natively Permeate, Record, and Stimulate Intracellular Activity in Neuronal and Cardiac Networks”. In: *Advanced Functional Materials* 32.8 (2022), p. 2108378. DOI: https://doi.org/10.1002/adfm.202108378. eprint: https://advanced.onlinelibrary.wiley.com/doi/pdf/10.1002/adfm.202108378. URL: https://advanced.onlinelibrary.wiley.com/doi/abs/10.1002/adfm.202108378.

[63] Yaoyao Liu et al. “Nanomaterial-based microelectrode arrays for in vitro bidirectional brain–computer interfaces: a review”. In: *Microsystems & Nanoengineering* 9.1 (2023), p. 13.

[64] John F. Barry et al. “Optical magnetic detection of single-neuron action potentials using quantum defects in diamond”. In: *Proceedings of the National Academy of Sciences* 113.49 (2016), pp. 14133–14138. DOI: 10.1073/pnas.1601513113. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1601513113. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1601513113.

[65] Madhur Parashar, Kasturi Saha, and Sharba Bandyopadhyay. “Axon hillock currents enable single-neuron-resolved 3D reconstruction using diamond nitrogen-vacancy magnetometry”. In: *Communications Physics* 3.1 (Oct. 2020), p. 174. DOI: 10.1038/s42005-020-00439-6. URL: https://doi.org/10.1038/s42005-020-00439-6.

[66] Pierre Hohenberg and Walter Kohn. “Inhomogeneous electron gas”. In: *Physical review* 136.3B (1964), B864.

[67] G Vignale and Mark Rasolt. “Density-functional theory in strong magnetic fields”. In: *Physical review letters* 59.20 (1987), p. 2360.

[68] Frank De Groot. “High-resolution X-ray emission and X-ray absorption spectroscopy”. In: *Chemical reviews* 101.6 (2001), pp. 1779–1808.

[69] PRGDJ Graves and D Gardiner. “Practical raman spectroscopy”. In: *Springer* 10 (1989), pp. 978–3.

[70] M Zahid Hasan and Charles L Kane. “Colloquium: topological insulators”. In: *Reviews of modern physics* 82.4 (2010), pp. 3045–3067.

[71] Charles L Kane and Eugene J Mele. “Quantum spin Hall effect in graphene”. In: *Physical review letters* 95.22 (2005), p. 226801.

[72] Charles L Kane and Eugene J Mele. “Z 2 topological order and the quantum spin Hall effect”. In: *Physical review letters* 95.14 (2005), p. 146802.

[73] Liang Fu, Charles L Kane, and Eugene J Mele. “Topological insulators in three dimensions”. In: *Physical review letters* 98.10 (2007), p. 106803.

[74] Markus Konig et al. “Quantum spin Hall insulator state in HgTe quantum wells”. In: *Science* 318.5851 (2007), pp. 766–770.

[75] David Hsieh et al. “A tunable topological insulator in the spin helical Dirac transport regime”. In: *Nature* 460.7259 (2009), pp. 1101–1105.

[76] A Yu Kitaev. “Fault-tolerant quantum computation by anyons”. In: *Annals of physics* 303.1 (2003), pp. 2–30.

[77] A Yu Kitaev. “Quantum error correction with imperfect gates”. In: *Quantum communication, computing, and measurement*. Springer, 1997, pp. 181–188.

[78] Eric Dennis et al. “Topological quantum memory”. In: *Journal of Mathematical Physics* 43.9 (2002), pp. 4452–4505.

[79] Nissim Ofek et al. “Extending the lifetime of a quantum bit with error correction in superconducting circuits”. In: *Nature* 536.7617 (2016), pp. 441–445.

[80] Google Quantum AI. “Suppressing quantum errors by scaling a surface code logical qubit”. In: *Nature* 614.7949 (2023), pp. 676–681.

[81] Mitsuki Katsuda, Kosuke Mitarai, and Keisuke Fujii. “Simulation and performance analysis of quantum error correction with a rotated surface code under a realistic noise model”. In: *Physical Review Research* 6.1 (2024), p. 013024.

[82] Zeyuan Zhou, Andrew Ji, and Yongshan Ding. “Surface Code Error Correction with Crosstalk Noise”. In: *arXiv preprint arXiv:2503.04642* (2025).

[83] Jens Koch et al. “Charge-insensitive qubit design derived from the Cooper pair box”. In: *Physical Review A—Atomic, Molecular, and Optical Physics* 76.4 (2007), p. 042319.

[84] Frank Arute et al. “Quantum supremacy using a programmable superconducting processor”. In: *Nature* 574.7779 (2019), pp. 505–510.

[85] M Veldhorst et al. “An addressable quantum dot qubit with fault-tolerant control-fidelity”. In: *Nature nanotechnology* 9.12 (2014), pp. 981–985.

[86] Alexander PM Place et al. “New material platform for superconducting transmon qubits with coherence times exceeding 0.3 milliseconds”. In: *Nature communications* 12.1 (2021), p. 1779.

[87] 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 (2018), pp. 102–106.

[88] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

[89] Jesse J Langille and Richard E Brown. “The synaptic theory of memory: a historical survey and reconciliation of recent opposition”. In: *Frontiers in systems neuroscience* 12 (2018), p. 52.

[90] Alex M Thomson and Jim Deuchars. “Temporal and spatial properties of local circuits in neocortex”. In: *Trends in neurosciences* 17.3 (1994), pp. 119–126.

[91] Henry Markram et al. “Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs”. In: *Science* 275.5297 (1997), pp. 213–215.

[92] Misha Tsodyks, Klaus Pawelzik, and Henry Markram. “Neural networks with dynamic synapses”. In: *Neural computation* 10.4 (1998), pp. 821–835.

[93] Natalia Caporale and Yang Dan. “Spike timing–dependent plasticity: a Hebbian learning rule”. In: *Annu. Rev. Neurosci.* 31.1 (2008), pp. 25–46.

[94] Keith T Butler et al. “Machine learning for molecular and materials science”. In: *Nature* 559.7715 (2018), pp. 547–555.

[95] Giuseppe Carleo et al. “Machine learning and the physical sciences”. In: *Reviews of Modern Physics* 91.4 (2019), p. 045002.

[96] Keith A Brown et al. “Machine learning in nanoscience: big data at small scales”. In: *Nano Letters* 20.1 (2019), pp. 2–10.

[97] John Jumper et al. “Highly accurate protein structure prediction with AlphaFold”. In: *nature* 596.7873 (2021), pp. 583–589.

[98] *Nobel prize in chemistry 2024*. https://www.nobelprize.org/prizes/chemistry/2024/press-release/. Accessed: 2025-06-04.

[99] Jörg Behler and Michele Parrinello. “Generalized neural-network representation of high-dimensional potential-energy surfaces”. In: *Physical review letters* 98.14 (2007), p. 146401.

[100] Stefan Chmiela et al. “Towards exact molecular dynamics simulations with machine-learned force fields”. In: *Nature communications* 9.1 (2018), p. 3887.

[101] Oliver T Unke et al. “Machine learning force fields”. In: *Chemical Reviews* 121.16 (2021), pp. 10142–10186.

[102] Giuseppe Carleo and Matthias Troyer. “Solving the quantum many-body problem with artificial neural networks”. In: *Science* 355.6325 (2017), pp. 602–606.

[103] Matija Medvidović and Javier Robledo Moreno. “Neural-network quantum states for many-body physics”. In: *The European Physical Journal Plus* 139.7 (2024), pp. 1–26.

[104] Román Orús. “Tensor networks for complex quantum systems”. In: *Nature Reviews Physics* 1.9 (2019), pp. 538–550.

[105] David Ceperley and Berni Alder. “Quantum monte carlo”. In: *Science* 231.4738 (1986), pp. 555–560.

[106] William MC Foulkes et al. “Quantum Monte Carlo simulations of solids”. In: *Reviews of Modern Physics* 73.1 (2001), p. 33.

[107] Luciano Loris Viteritti, Riccardo Rende, and Federico Becca. “Transformer variational wave functions for frustrated quantum spin systems”. In: *Physical Review Letters* 130.23 (2023), p. 236401.

[108] Riccardo Rende et al. “A simple linear algebra identity to optimize large-scale neural network quantum states”. In: *Communications Physics* 7.1 (2024), p. 260.

[109] Eliska Greplova et al. “Fully automated identification of two-dimensional material samples”. In: *Physical Review Applied* 13.6 (2020), p. 064017.

[110] Axel UJ Lode et al. “Optimized observable readout from single-shot images of ultracold atoms via machine learning”. In: *Physical Review A* 104.4 (2021), p. L041301.

[111] Sandesh S Kalantre et al. “Machine learning techniques for state recognition and auto-tuning in quantum dots”. In: *npj Quantum Information* 5.1 (2019), p. 6.

[112] Renato Durrer et al. “Automated tuning of double quantum dots into specific charge states using neural networks”. In: *Physical Review Applied* 13.5 (2020), p. 054019.

[113] Hyungil Moon et al. “Machine learning enables completely automatic tuning of a quantum device faster than human experts”. In: *Nature communications* 11.1 (2020), p. 4161.

[114] Agnes Valenti et al. “Scalable Hamiltonian learning for large-scale out-of-equilibrium quantum dynamics”. In: *Physical Review A* 105.2 (2022), p. 023302.

[115] Frederik Wilde et al. “Scalably learning quantum many-body Hamiltonians from dynamical data”. In: *arXiv preprint arXiv:2209.14328* (2022).

[116] Kris Tucker et al. “Hamiltonian learning using machine-learning models trained with continuous measurements”. In: *Physical Review Applied* 22.4 (2024), p. 044080.

[117] Anna Dawid et al. “Modern applications of machine learning in quantum sciences”. In: *arXiv preprint arXiv:2204.04198* (2022).

[118] Elad Schneidman et al. “Weak pairwise correlations imply strongly correlated network states in a neural population”. In: *Nature* 440.7087 (2006), pp. 1007–1012.

[119] Gašper Tkačik et al. “The simplest maximum entropy model for collective behavior in a neural network”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2013.03 (2013), P03011.

[120] Johanna Senk et al. “Connectivity concepts in neuronal network modeling”. In: *PLoS Computational Biology* 18.9 (2022), e1010086.

[121] Lane McIntosh et al. “Deep learning models of the retinal response to natural scenes”. In: *Advances in neural information processing systems* 29 (2016).

[122] Guillaume Bellec et al. “Fitting summary statistics of neural data with a differentiable spiking network simulator”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 18552–18563.

[123] Trung Le and Eli Shlizerman. “Stndt: Modeling neural population activity with spatiotemporal transformers”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 17926–17939.

[124] Demis Hassabis et al. “Neuroscience-inspired artificial intelligence”. In: *Neuron* 95.2 (2017), pp. 245–258.

[125] Colin J Brown and Ghassan Hamarneh. “Machine learning on human connectome data from MRI”. In: *arXiv preprint arXiv:1611.08699* (2016).

[126] Tabinda Sarwar et al. “Structure-function coupling in the human connectome: A machine learning approach”. In: *NeuroImage* 226 (2021), p. 117609.

[127] Michał Januszewski et al. “High-precision automated reconstruction of neurons with flood-filling networks”. In: *Nature methods* 15.8 (2018), pp. 605–610.

[128] Pierre Yger et al. “A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo”. In: *elife* 7 (2018), e34518.

[129] Ramin Toosi, Mohammad Ali Akhaee, and Mohammad-Reza A Dehaqani. “An automatic spike sorting algorithm based on adaptive spike detection and a mixture of skew-t distributions”. In: *Scientific Reports* 11.1 (2021), p. 13925.

2

BACKGROUND

This chapter presents the essential background for the topics explored in the thesis, providing a brief, high-level introduction to varied domains, aiming to cover just enough to fill any foundational knowledge gaps needed for the subsequent chapters. We start with Artificial Intelligence basics, covering the evolution of neural networks and the rise of deep generative models. Then, we explore the simulation and analysis of neural activity, from biophysical models to machine learning approaches. The discussion then moves to Quantum Computing, covering qubits, quantum gates, and circuits, with a focus on spin qubits and their experimental tuning using charge stability diagrams. Another section bridges some of these fields by introducing Quantum Machine Learning, with a focus on quantum generative models, like the Quantum Generative Adversarial Network. Finally, we introduce Neural Quantum States, exploring how neural networks can be used as variational ansatzes to find ground states of quantum systems.

2.1. ARTIFICIAL INTELLIGENCE BASICS

2.1.1. FROM NEURONS TO DEEP LEARNING

While AI does not have an unique definition, within this thesis we will refer to Artificial Intelligence (AI) as the set of systems used to tackle tasks that would require some level of human intelligence to be solved [1]. One the main sub-fields of AI is machine learning (ML), in which algorithms iteratively improve their performance by learning from data instead of using handcrafted rules [2]. Before the 2000s and the advent of the deep learning era [3], main ML approaches included techniques like linear regression, decision trees, and support vector machines. Linear regression, in which a function $f(x) = \mathbf{w}^T\mathbf{x} + b$ is used to fit data, by adjusting learnable parameters called weights (a vector $\mathbf{w}$) and a bias (constant b), so that the function minimizes the mean-square error between predicted outputs and the correct values from the data [4]. Decision trees, in which the input/feature space is recursively partitioned into regions with same output/target values, by selecting thresholds that maximize information gain [5]. Support vector machines (SVM), in which the goal consists of finding boundaries or hyperplanes in high-dimensional spaces that separate the input space in different classes by maximizing a margin (distance between closest points of each class) [6].

Another class of models in machine learning, Neural Networks (NN), were developed in the mid 20th century, but saw exponential rise in usage, accuracy, and development, in the 2000s. NNs are composed of individual units called perceptrons [7], which functioning was inspired by the biological neuron [8] and the mathematical model for a neuron [9]. The perceptron computes the weighted sum $z = \sum_i w_i x_i + b$ of inputs $\mathbf{x}$, and return $\sigma(z)$ as output, where σ is a non-linear activation function, and $\mathbf{w}$ and b are learnable parameters, called weights and biases. When perceptrons/neurons are organized in layers - usually one input layer, one or more hidden layers, and one output layer - the architecture is called a multi-layer perceptron (MLP), which can be used to approximate any function with arbitrary accuracy, as proven in the universal approximation theorem [10]. Even though MLPs can approximate any function, they need to be trained so that the optimal parameters θ (which include both the $\mathbf{w}$ and b defined before) are learned, and this is not necessarily an easy task, especially when the networks are deep (many hidden layers). This changed when an algorithm called backpropagation, which efficiently computes gradients of a loss function $\mathscr{L}(\theta)$ with respect to the parameters θ, was developed in 1986 [11]. In the late 1990s and early 2000s, progress in neural networks research rapidly advanced, reaching a milestone in 2012, when a neural network architecture called AlexNet achieved unseen accuracy in classifying images from a large image database [12], starting the deep learning era [3].

2.1.2. MACHINE LEARNING PARADIGMS AND MAIN ARCHITECTURES

Broadly speaking, ML methods can be categorized into three main classes: supervised learning, unsupervised learning, and reinforcement learning [13]. In supervised learning, models learn tasks by being trained on labeled data so that they can generalize and be used to perform the task on unseen data [14]. In this setting, the loss function, the object to be minimized that dictates how the network's learnable parameters are update, consists of some functional form of the error between the network's prediction/output,

and the real output - the labeled data sample. This scheme is very efficient, but it requires a large amount of labeled data, that grows with the size of network/ number of trainable parameters, which can be necessary to solve more complex problems [15]. On the other end of the "supervision" spectrum, we have unsupervised learning, where the network has no access to labeled data. In this case, the goal is usually to learn general patterns present in the data, so that it can be exploited to predict classes or values of unseen samples. Common techniques used in the unsupervised setting include clustering and dimensionality reduction [16, 17, 18, 19, 20]. The third category, reinforcement learning (RL), is fundamentally different than the cases already described [21, 22]. In RL, an agent learns to take actions in a certain environment, so to maximize an cumulative reward. The method emulates more closely the psychology of learning, with inspirations coming from studies on learning in animals and children. Instead of having a predefined dataset, the agent has access to an environment, which changes depending on the actions it takes. A reward function is design to incentivize the agent to take actions that navigate or change the environment to achieve a goal. RL is particularly useful when one wants to build an agent to simulate a task performed by a human, with some of the most successful examples are videogames [23, 24].

Different architectures, deviating from the simple case of the MLP, can be designed to more efficiently solve specific tasks, exploiting symmetries and structures in the data. In Convolutional Neural Networks (CNNs) a specific operation called convolution is used instead of the usual weighted sum present in MLPs, so that the local correlations present in image data is better exploited, making them a better option in tasks that operate on images [25], like segmentation (separating individual objects present in an image) [26], reconstructing missing parts of pictures [27], or denoising noisy images [28]. Recurrent Neural Network (RNNs) are another example of an architecture tailored to work in specific settings [29, 30]. RNNs, and its variants like LSTMs [31] and GRUs [32], implement self-feedback connections, which enable the processing of sequential data, such as time series [33, 34] or language [35, 36]. More recently, mainly in the field of natural language processing, recurrent layers were replaced by an operation called attention, which is the base of the transformer architecture [37]. In transformers, instead of having a word in a sentence being related to the previous word, the following word, and to themselves, like in the recurrent case, a position-dependent correlation between all words in the input is calculated. The advent of transformers completely changed the field of NLP at first, and then most of the ML fields, achieving state of the art results across different modalities and benchmarks [38, 39, 40].

2.1.3. GENERATIVE LEARNING, AUTOENCODERS, AND DIFFUSION MODELS

Many of the architectures described up to this point in the thesis are primarily used for discriminative tasks, like classification or regression, in which the goal is to predict a label or a numerical value, given an input. Even in the case of unsupervised learning, where the data in unlabeled, very often the goal of the learning algorithm is still discriminative: learning some structure of the data so that new samples can be classified or regressed. However, there is a specific subclass of unsupervised learning known as generative learning. In generative learning, the goal is to learn the underlying distribu-

tion $p_D(x)$ of a dataset D, so that new samples can be drawn from it. Some of the most influential paradigms in recent generative learning literature include variational autoencoders, generative adversarial networks, and more recently, diffusion models.

Variational Auto-Encoders (VAEs) [41] build upon the idea behind a (standard) Auto-Encoder [42, 43, 44], which compresses input data $\mathbf{x}$ into a lower dimensional space $\mathbf{z}$ using an encoder module, and then uses a decoder to try to reconstruct the input from the latent space $\mathbf{z}$. VAEs introduce a probabilistic component, where the encoder maps inputs to a distribution in latent space $q(\mathbf{z}|\mathbf{x})$, and the decoder learns the likelihood $p(\mathbf{x}|\mathbf{z})$, modeling the probability of seeing a sample $\mathbf{x}$ given a latent variable $\mathbf{z}$. Training is performed using a loss function that comprises both a reconstruction term, calculating the difference between the reconstructed output and the original input, and a KL divergence term that enforces the latent space to be a normal distribution [45, 46].

Generative Adversarial Networks (GANs) [47] propose a different approach, based on game theory: two networks, a generator $G(\mathbf{z})$ and a discriminator $D(\mathbf{x})$ are trained simultaneously and in a competitive scheme. The generator is responsible to map noise to data samples, or in other words, to produce synthetic samples similar to the real ones; the discriminator attempts to distinguishing real data from synthetic samples. This behavior is described by the min-max equation:

$$\min_G \max_D E_{x\sim P_x}\left[\log D(x)\right] + E_{z\sim P_z}\left[\log(1 - D(G(z)))\right]. \tag{2.1}$$

While very unstable and difficult to train when they were first proposed, GANs and its variants, like the Wasserstein GAN [48, 49], are able to produce extremely realistic samples. This generative power is mainly exploited in computer vision, in which GANs can be used to generate high-quality synthetic images [50, 51].

More recently, Diffusion Models (DM) [52, 53] have risen as a more stable, scalable, and efficient alternative to GANs. These models learn to reverse a gradual noising process: starting from a data sample $\mathbf{x}_0$, Gaussian noise is added over T steps by using a forward diffusion process $q(\mathbf{x}_t|\mathbf{x}_{t-1})$. A neural network is then used to learn the inverse process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, effectively learning to denoise the data. When trained with a large amount of image data, and by improving parts of the diffusion scheme, DM can be used to generate diverse and hyper-realistic images [54, 55]. They can be made conditioned, so that the image output depends on the initial random noise, plus some other constraint, from text, which effectively create models that generate images from sentences, to other (parts of) images, which can be used to inpaint or upscale images [56, 57, 58, 59].

2.2. SIMULATION AND ANALYSIS OF NEURAL ACTIVITY

2.2.1. BIOPHYSICAL NEURONS

The brain consists of a highly interconnected network of billions of neurons. While artificial neural networks, as described in the previous section, are inspired by the biological brain, and can be thought on a very high-level as an abstraction of it, the simulation of neuronal activity in biological networks is way more complex, and a correct description of its components requires tools from biology and physics. The pioneering Hodgkin-Huxley (HH) describes the generation and propagation of action potential in neurons

by combining ideas from physics and biology: physics by modeling the potential of a cell membrane using partial differential equations from electrodynamics, and biology to describe the ion channels and their interaction [9]. Although biophysically precise, the HH model is computationally expensive for large-scale network simulations. Simplified alternatives include integrate-and-fire models, like leaky integrate and fire (LIF) [60], which neglects the ion channel details, and only considers that the neurons integrate over inputs, and spike/fire after reaching a certain threshold. This simplification allows the efficient simulation of networks with many more neurons when compared to the more precise HH model, still showing high accuracy when describing phenomena like synchronization and information propagation in cortical and sub-cortical neural circuits [61, 62].

2.2.2. SPIKE TRAINS: MAXIMUM ENTROPY MODELS AND MACHINE LEARNING

Moving to a higher level of abstraction, we can think of neurons as units that communicate using short electrical impulses known as spikes, with a spike train being the time trace of the spike of a single neuron, or, of a whole sub-network when activity from several connected neurons is recorded at the same time. This simplifies neuronal activity so that neurons can only assume two possible states over a certain time bin: active (1) or silent (0) [63, 64].

Rather than trying to model the correct biophysics of neuronal dynamics, one can focus on the description of the statistics of spike trains [65, 66]. One technique that uses tools from statistical physics to successfully capture the complex statistics of spike trains is Maximum-Entropy (MaxEnt) models [67, 68], which given some spike train as data, aim to find the most unbiased (highest entropy) probability distribution that matches certain properties in the data, like average firing rate (how often neurons spike in a given time) and pairwise correlation (how correlated are spikes between pairs of neurons). This allows to model spike trains without assuming how they are generated. First results showed that, for small populations of the salamander retina ganglion cells, a model considering only pairwise correlations between neurons captures most of the significant statistics, suggesting that higher-order correlations were not needed to efficiently describe complex spike trains [67]. The finding was reinforced when studying larger networks of different types of neurons, but with small deviations from the correct statistics starting to emerge [69, 70, 71, 72, 73]. Refined version of the model include higher order correlations, correction terms, and temporal dependence, making it possible to capture more complex statistical behaviors, like synchronization and bursting patterns [74, 75, 76].

An even more simplified alternative to MaxEnt models, that discards the physical-informed assumption that the probability distribution that fits the data needs to be the one that gives the highest entropy, is to use machine learning models to produce synthetic data that matches the statistics of a spike train dataset. Early works include a supervised approach, where a CNN is trained to predict spike trains dynamics given external stimula [77], however, it is clearly limited by the necessity of labeled data and its deterministic nature. Generative models offer a better alternative, given that they are designed to fit the distribution of the dataset. Generative approaches to generate syn-

thetic neuronal activity include using VAEs, RNNs, and GANs [78, 79, 80, 81]. Like in several other fields, transformers also showed great performance to model neuronal activity [82, 83, 84]. More recently, an approach using a diffusion model showed promising results, which, aside from generating realistic spike trains, encodes low-dimensional latent representations of the data [85].

In Chapter 4, we use concepts from this section and the previous one (2.1), to show how it possible to automate the analysis routine of MEA data by using convolutional neural networks.

2.3. Quantum Computing with Quantum Dots

2.3.1. Qubits, Gates, and Quantum Circuits

The basic unit in quantum computing is the qubit, a quantum object that can be in the state of superposition between its basis states, $|0\rangle$ and $|1\rangle$, fundamentally different than their classical version, the bit, which can only be in the binary states 0 or 1 [86]. In practice, the qubit is a quantum two-level system, and thus it needs to be experimentally implemented; common implementations of qubits include the spin of electrons, the polarization state of photons, and the discrete energy levels of an atom. The state of a qubit is changed by using gates, a unitary operation that coherently acts on one or more qubits. An example of a quantum gate is X, which in matrix form is described as

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

which, when applied to a qubit in a general superposition state

$$\psi = a|0\rangle + b|1\rangle\,,$$

returns the state $\psi = b|0\rangle + a|1\rangle$, flipping its components - so if a qubit is in the $|0\rangle$ state and X is applied, the qubit state changes to $|1\rangle$, and vice-versa. Different operations are defined by other gates, and small sets of gates can be combined to get universality: any unitary transformation of the qubits state can be obtained [87]. The act of sequentially applying gates to qubits, and therefore consecutively changing the system state, is equivalent to running what is known as a digital quantum algorithm, which can be represented graphically by a quantum circuit [88, 86]. Digital quantum algorithms, like the Shor's factoring algorithm [89], or Grover's search algorithm [90], make use of carefully chosen sequences of single and two qubit gates so that the overall unitary evolution of the quantum system reaches a speed-up when compared to classical alternatives.

2.3.2. Spin Qubits and Charge Stability Diagrams

As mentioned before, in practice the qubit needs to be physically implemented as a quantum system. There are multiple ways of achieving this, with different implementations giving rise to different paradigms of quantum computing, like superconducting quantum computing if qubits are implementing using superconductor electronic circuits [91], or trapped-ion quantum computing if the qubit is encoded as different electronic states of confined ions [92, 93].

Spin qubits are implemented by confining spins of semiconductors, and using the *up* and *down* possible states of the spin as the two-level system [94]. The Loss–DiVincenzo proposal employs the spin of a single electron trapped in a gate-defined quantum dot (QD) as a qubit, taking advantage of the long coherence times of electron spins in certain semiconductor heterostructures [95]. This allows to perform single qubit gates by coherently rotating the spin of an isolated quantum dot, and two qubit gates by making two neighboring quantum dots interact [96, 97, 98, 99, 100].

One of the key requirements that needs to be met in order to use quantum dots as qubits, is having each quantum dot in an operational regime, which typically involves precise control over the number of electrons confined in each dot. This control is achieved using electrostatic gates that locally tune the potential landscape in the semiconductor and define the quantum dot itself. By adjusting the voltages applied to these gates, one can control both the formation of the dot and the number of electrons it holds [95, 101]. Charge stability diagrams (CSDs) are graphical representations that assist in this tuning process by mapping the electron occupation in each dot as a function of the applied gate voltages [102, 103]. Experimentally, charge stability diagrams are obtained by sweeping gate voltages over a two-dimensional grid and recording an electrical signal that reveals transitions in the dot's charge state. The two main ways of getting a CSD are: direct current (DC) transport measurements, in which the measured signal is the current through the dot under a small source–drain bias [104, 105]; and radio-frequency (RF) reflectometry, where the dot is embedded in a resonant circuit, and changes in the RF signal, caused by variations in the dot's charge configuration, are used to detect single-electron tunneling events [106, 107, 108].

2.3.3. OPERATING QUANTUM DOTS

The ability to record signals from quantum dots and construct charge stability diagrams is necessary, but not enough to set the system in an operational qubit state: one has to cleverly navigate the high-dimensional voltage landscape to calibrate each physical parameters. The process occurs in sequential stages, starting with coarse tuning, in which the device is cooled, sensors are activated, and voltages are swept to find wether the system is in a zero, one, or double dot configuration. Afterwards, virtual gates can be defined to mitigate crosstalk effects and allow control of individual dots. Next, in the charge state tuning stage, gate voltages are finely tuned to achieve exact number of electrons in each QD. This is one of the main steps in which the CSD can be analyzed and navigated (changing the voltages) to obtain target charge states - since crossing the lines in CSDs correspondent to changing the number of electrons in the QDs. Finally, some of the system's physical parameters are adjusted to obtain optimal control for quantum computation.

Manually tuning a dot by going through all the steps described is extremely labor-intensive, does not scale efficiently with the number of QDs/voltage gates, and requires repeating the task for different devices. In this context, classical optimization methods are used to automate the tuning [109, 110, 111]. While obtaining satisfactory results in some settings, the noisy nature of CSDs makes it challenging for these methods to precisely detect transition lines [112, 113]. As a more resilient-to-noise alternative, machine learning algorithms have been implemented to automate the tuning process, with dif-

ferent works addressing all steps of tuning, from coarse tuning [114, 115, 116], to setting virtual gates [117], and charge tuning [118, 119, 120]. We are now at the point that a combination of different techniques can be implemented to fully automate the tuning process [121, 122], with remaining challenges to refine the tuning routines, such as the need for standardized benchmarks and datasets to develop and test new algorithms, being addressed by the community [123].

In chapter 5 and 6, we use concepts from this section and the section 2.1, by proposing a CSD simulator, useful to efficiently build synthetic datasets for ML applications (Chapter 5), and in Chapter 6 by designing a ML approach to classify experimental CSDs, which can be used to build datasets with experimental measurement, and preliminary results on using diffusion models to complete measurements in CSDs, in order to reduce measurement time.

2.4. MACHINE LEARNING ON QUANTUM COMPUTERS

2.4.1. QUANTUM MACHINE LEARNING

One the possible intersections of machine learning models, as described in section 2.1, with quantum computing and the quantum circuit paradigm, introduced in section 2.3, is quantum machine learning (QML): running learning algorithms on quantum circuits, harnessing quantum phenomena like superposition and entanglement [124, 125, 126]. One standard model used in QML is the parametrized quantum circuit (PQC): quantum circuits with layers of parameter-dependent single-qubit rotations and entangling gates, which parameters are optimized classically based on measurement outcomes [127]. Some of the main architectures in machine learning were translated to the quantum regime, with examples such as Quantum Convolutional Networks [128], Quantum Recurrent Networks [129], and Quantum Transformers [130]; and several works study the application of PQCs, from classification tasks in toy problems to real classical and quantum data [131, 132, 133]. Nonetheless, several challenges remain in the field, from proving advantages compared to classical alternatives [134, 135], to finding an efficient training scheme [136, 137, 138] and dealing with vanishing gradients (Barren plateaus) [139, 140].

2.4.2. QUANTUM GENERATIVE LEARNING

Mirroring the categories in machine learning, QML can also be divided in different categories. When models are used to learn the distribution of a dataset, we have Quantum Generative Learning. In quantum generative learning, models prepare parameterized quantum states and generate samples via measurement [141]. When compared to discriminative tasks, quantum generative learning was shown to be resilient to training problems like Barren plateaus [142, 143]. Quantum Circuit Born Machines (QCBMs), for instance, use PQCs to encode distributions over bit strings, exploiting natural quantum randomness and entanglement, and show potential to encode distributions with complexity beyond classical models [144, 145]. Quantum Boltzmann Machines [146] and Quantum Variational Autoencoders [147] extend classical generative frameworks into the quantum domain. Quantum Generative Adversarial Networks (QGANs) are another example of a quantum adaptation of a classical architecture. They use the same scheme described in Section 2.1, implementing two networks, a generator and a dis-

criminator, but using a PQC for one of the networks (or both) [148]. Several works show that QGANs can efficiently learn classical distributions [149, 150, 151]. Moreover, refinements in QGAN architectures have advanced their scalability and performance: patch QGANs [152], which decompose high-dimensional data into smaller patches processed by compact PQC sub-generators, enables efficient image generation on limited qubit hardware, while a Wasserstein style GAN implementation, with a critic replacing the discriminator, allows QGANs to produce high-resolution images [153].

In Chapter 3, we use concepts from this section, and section 2.2, to implement an application of quantum generative learning in neuroscience, by using QGANs to simulate neuronal activity.

2.5. NEURAL QUANTUM STATES

2.5.1. VARIATIONAL GROUND STATES

One of the main challenges in modern science, and especially in quantum physics, is finding the ground states of quantum systems: the state with the lowest energy possible. This is a particularly important state that can be used to describe general properties of the system. For example, in quantum chemistry, the ground state of a molecule provides essential information about the system's stability, reactivity, and other chemical properties [154, 155]. For simple systems, it is feasible to exactly find the ground state; however, increasing the number of particles in the systems accompanies an exponential scaling of the Hilbert space, which make exact solutions impractical, and approximations methods are required. This computational complexity is one of the main reasons why quantum computers were proposed in the first place: simulating quantum systems efficiently [156, 157]. However, in the absence of scalable quantum hardware, alternative approaches based on classical variational methods have proven useful. The core idea behind variational methods is to propose an approximate wave function, known as an ansatz, which depends on a set of adjustable parameters and, according to the variational principle, the expectation value of the system Hamiltonian for the approximated wave function will always be higher than the true ground state energy [158]. This principle can be exploited to reframe the ground state problem as an optimization problem: the expected value of the system Hamiltonian is minimized with respect to the ansatz parameters, and the process is iterated over until the approximate energy converges to the true ground state energy. Several variational methods can be used, each with their own advantages and limitations. For instance, tensor networks methods [159], such as matrix product states and projected entangled pair states, in which the wave function is encoded in a contracted tensor of a network of individual tensors, enable obtaining high-accuracy ground state approximations with efficient computational cost, but lose most of their strength when dealing with system in higher-dimensional, or that have volume-law entanglement [160]. Quantum Monte Carlo methods [161], such as Diffusion Monte Carlo, which uses a Green's function to calculate the low energy states of the system Hamiltonian, show state of the art accuracy for certain cases, but are affected by the sign problem when simulating fermionic or frustrated systems [162]. The advent of deep learning, with neural networks proven to be universal function approximators [163], led to idea of using machine learning models as variational ansatzes to approx-

imate quantum wave functions. This approach, called Neural Quantum States (NQS) [164], in which the ansatz parameters to be optimized are the weights and biases of a neural network, shown extremely successful in simulating quantum states, even in complex cases such as highly-entangled or frustrated systems [165].

2.5.2. NQS: VARIATIONAL AND FULL-SUM STATES

Using a neural networks as an ansatz for the variational problem does not solve the curse of dimensionality: the energy of the system must be assessed in order to be minimized, and to obtain the exact energy, one needs to calculate the expectation value of the Hamiltonian with respect to all basis states of the wave function, which grow exponentially with the system size. This is circumvented, like in other variational methods, by using techniques like Markov Chain Monte Carlo to sample a polynomial number of significant configurations to calculate an approximation of the system's energy. This makes the computation of NQS efficient even when simulating larger systems, at the cost of introducing noise and biases associated with the sampling routine. The process of summing over all the (2^N for N particles) possible configurations is called "full sum state". While computationally inefficient and unfeasible for large systems, minimizing the exact energy by summing over all possible configurations, is often used in theoretical works where the goal is to study general properties of NQS that do not originate from the sampling method.

2.5.3. THE RESTRICTED BOLTZMANN MACHINE

Among all neural networks architectures, the Restricted Boltzmann Machine (RBM) [166] stands out in the field of NQS, being the first ansatz used in the seminal work by Carleo and Troyer [164], and is still being studied and implemented in recent works, both in the neural quantum state community [167, 168] and in the machine learning community [169]. Originally proposed in 1986 by Paul Smolensky [170], and later popularized by Geoffrey Hinton and collaborators [171], the RBM is a type of generative neural network with two layers: a visible layer, which in NQS represents the physical degree of freedom of the quantum system, and a hidden layer, which encodes a non-linear transformation of the visible layer. The RBM, and its deep learning version, the deep belief network [172], were some of the first generative models to show useful applications, from early works using it for dimensionality reduction [171] and feature learning [173], where training methods used sampling techniques and algorithms like contrastive divergence [174], and later in tailored applications such as quantum physics, using the network in a more discriminative manner and using the standard backpropagation algorithm to train it [165].

2.5.4. FINDING PHYSICS IN NEURAL NETWORKS

One reason why restricted Boltzmann machines were widely accepted in the physics community is the possibility of obtaining the model's energy in an analytical form, which makes it possible to physically investigate the optimization process of the network. Another exciting aspect of using RBMs as ansatz in variational methods is the potential to extract physical insights from the network learned parameters [175]. The weights connecting visible and hidden layers can be directly related with physical correlations in

the quantum system, offering an extra framework to interpret what the model is learning. This analytical form can even be exploited to represent transformations between quantum states as transformation in the RBM parameters, which can be use to exactly implement quantum gates in the classical simulation of quantum circuits [176]. However, this analytical form does not always hold for more complex and deeper architecture, like convolution or recurrent networks. Nonetheless, a growing body of literature is now dedicated to applying interpretability techniques to extract physical representations from networks used to solve physics problems [177, 178, 179]. Some recent works show promising results in uncovering physical quantities encoded in the parameters of NQS ansatzes [180], even in cases where the networks are more complex than RBMs, including transformer-based architectures [181, 182].

In Chapter 7 we show how it is possible to detect phase transitions in quantum systems, by approximating the wave function at different points of the phase diagram with NQS, and mapping the networks' weights to a lower dimensional space, using Principal Component Analysis.

REFERENCES

[1] Haroon Sheikh, Corien Prins, and Erik Schrijvers. "Artificial intelligence: definition and background". In: *Mission AI: The new system technology*. Springer, 2023, pp. 15–41.

[2] Michael I Jordan and Tom M Mitchell. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245 (2015), pp. 255–260.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[4] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[5] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1 (1986), pp. 81–106.

[6] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20 (1995), pp. 273–297.

[7] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[8] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[9] Alan L Hodgkin and Andrew F Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4 (1952), p. 500.

[10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[13] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[14] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[15] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *arXiv preprint arXiv:1611.03530* (2016).

[16] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. "Data clustering: a review". In: *ACM computing surveys (CSUR)* 31.3 (1999), pp. 264–323.

[17] Lior Rokach and Oded Maimon. "Clustering methods". In: *Data mining and knowledge discovery handbook* (2005), pp. 321–352.

[18] J Teenbaum, DD Silva, and JA Langford. “global geometric framework for nonlinear dimensionality reduction [J]”. In: *Science* 290.5500 (2000), pp. 2319–2323.

[19] Ian T Jolliffe and Jorge Cadima. “Principal component analysis: a review and recent developments”. In: *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), p. 20150202.

[20] Felipe L Gewers et al. “Principal component analysis: A natural approach to data exploration”. In: *ACM Computing Surveys (CSUR)* 54.4 (2021), pp. 1–34.

[21] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[22] Ashish Kumar Shakya, Gopinatha Pillai, and Sohom Chakrabarty. “Reinforcement learning algorithms: A brief survey”. In: *Expert Systems with Applications* 231 (2023), p. 120495.

[23] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[24] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[25] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[26] 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: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241.

[27] Gabriel Eilertsen et al. “HDR image reconstruction from a single exposure using deep CNNs”. In: *ACM transactions on graphics (TOG)* 36.6 (2017), pp. 1–15.

[28] Kai Zhang et al. “Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising”. In: *IEEE transactions on image processing* 26.7 (2017), pp. 3142–3155.

[29] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. *Learning internal representations by error propagation*. 1985.

[30] Michael I Jordan. “Serial order: A parallel distributed processing approach”. In: *Advances in psychology*. Vol. 121. Elsevier, 1997, pp. 471–495.

[31] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[32] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[33] Daan Wierstra, Jürgen Schmidhuber, and Faustino J Gomez. “Evolino: Hybrid neuroevolution/optimal linear search for sequence learning”. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), Edinburgh*. 2005, pp. 853–858.

[34] Pankaj Malhotra et al. “Long short term memory networks for anomaly detection in time series”. In: *Proceedings*. Vol. 89. 9. 2015, p. 94.

[35] Alex Graves and Jürgen Schmidhuber. “Offline handwriting recognition with multidimensional recurrent neural networks”. In: *Advances in neural information processing systems* 21 (2008).

[36] Shujie Liu et al. “A recursive recurrent neural network for statistical machine translation”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2014, pp. 1491–1500.

[37] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[38] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).

[39] Thomas Wolf et al. “Transformers: State-of-the-art natural language processing”. In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 2020, pp. 38–45.

[40] Qingsong Wen et al. “Transformers in time series: A survey”. In: *arXiv preprint arXiv:2202.07125* (2022).

[41] Diederik P Kingma, Max Welling, et al. *Auto-encoding variational bayes*. 2013.

[42] Pierre Baldi and Kurt Hornik. “Neural networks and principal component analysis: Learning from examples without local minima”. In: *Neural networks* 2.1 (1989), pp. 53–58.

[43] Erkki Oja. “Simplified neuron model as a principal component analyzer”. In: *Journal of mathematical biology* 15 (1982), pp. 267–273.

[44] Geoffrey E Hinton and Richard Zemel. “Autoencoders, minimum description length and Helmholtz free energy”. In: *Advances in neural information processing systems* 6 (1993).

[45] Yunchen Pu et al. “Variational autoencoder for deep learning of images, labels and captions”. In: *Advances in neural information processing systems* 29 (2016).

[46] Carl Doersch. “Tutorial on variational autoencoders”. In: *arXiv preprint arXiv:1606.05908* (2016).

[47] Ian J Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems* 27 (2014).

[48] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein generative adversarial networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 214–223.

[49] Ishaan Gulrajani et al. “Improved training of wasserstein gans”. In: *Advances in neural information processing systems* 30 (2017).

[50] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[51] Ian Goodfellow. “Nips 2016 tutorial: Generative adversarial networks”. In: *arXiv preprint arXiv:1701.00160* (2016).

[52] Jascha Sohl-Dickstein et al. “Deep unsupervised learning using nonequilibrium thermodynamics”. In: *International conference on machine learning*. pmlr. 2015, pp. 2256–2265.

[53] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[54] Alexander Quinn Nichol and Prafulla Dhariwal. “Improved denoising diffusion probabilistic models”. In: *International conference on machine learning*. PMLR. 2021, pp. 8162–8171.

[55] Robin Rombach et al. “High-resolution image synthesis with latent diffusion models”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[56] Chitwan Saharia et al. “Palette: Image-to-image diffusion models”. In: *ACM SIGGRAPH 2022 conference proceedings*. 2022, pp. 1–10.

[57] Anji Liu, Mathias Niepert, and Guy Van den Broeck. “Image inpainting via tractable steering of diffusion models”. In: *arXiv preprint arXiv:2401.03349* (2023).

[58] Ciprian Corneanu, Raghudeep Gadde, and Aleix M Martinez. “Latentpaint: Image inpainting in latent space with diffusion models”. In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2024, pp. 4334–4343.

[59] Jianyi Wang et al. “Exploiting diffusion prior for real-world image super-resolution”. In: *International Journal of Computer Vision* 132.12 (2024), pp. 5929–5949.

[60] Larry F Abbott. “Lapicque’s introduction of the integrate-and-fire model neuron (1907)”. In: *Brain research bulletin* 50.5-6 (1999), pp. 303–304.

[61] Nicolas Brunel. “Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons”. In: *Journal of computational neuroscience* 8 (2000), pp. 183–208.

[62] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[63] Wulfram Gerstner and J van Hemmen. “How to describe neuronal activity: spikes, rates, or assemblies?” In: *Advances in neural information processing systems* 6 (1993).

[64] Christopher D Fiorillo, Jaekyung K Kim, and Su Z Hong. “The meaning of spikes from the neuron’s point of view: predictive homeostasis generates the appearance of randomness”. In: *Frontiers in Computational Neuroscience* 8 (2014), p. 49.

[65] William Bialek et al. “Reading a neural code”. In: *Advances in neural information processing systems* 2 (1989).

[66] Steven P Strong et al. “Entropy and information in neural spike trains”. In: *Physical review letters* 80.1 (1998), p. 197.

[67] Elad Schneidman et al. “Weak pairwise correlations imply strongly correlated network states in a neural population”. In: *Nature* 440.7087 (2006), pp. 1007–1012.

[68] Gasper Tkacik et al. “Ising models for networks of real neurons”. In: *arXiv preprint q-bio/0611072* (2006).

[69] Shan Yu et al. “A small world of neuronal synchrony”. In: *Cerebral cortex* 18.12 (2008), pp. 2891–2901.

[70] Aonan Tang et al. “A maximum entropy model applied to spatial and temporal correlations from cortical networks in vitro”. In: *Journal of Neuroscience* 28.2 (2008), pp. 505–518.

[71] Ifije E Ohiorhenuan et al. “Sparse coding and high-order correlations in fine-scale cortical networks”. In: *Nature* 466.7306 (2010), pp. 617–621.

[72] Olivier Marre et al. “Mapping a complete neural population in the retina”. In: *Journal of Neuroscience* 32.43 (2012), pp. 14859–14873.

[73] Jakob H Macke, Iain Murray, and Peter Latham. “How biased are maximum entropy models?” In: *Advances in neural information processing systems* 24 (2011).

[74] Takamitsu Watanabe et al. “A pairwise maximum entropy model accurately describes resting-state human brain networks”. In: *Nature communications* 4.1 (2013), p. 1370.

[75] Gašper Tkačik et al. “Searching for collective behavior in a large network of sensory neurons”. In: *PLoS computational biology* 10.1 (2014), e1003408.

[76] Hassan Nasser, Olivier Marre, and Bruno Cessac. “Spatio-temporal spike train analysis for large scale networks using the maximum entropy principle and Monte Carlo method”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2013.03 (2013), P03006.

[77] Lane McIntosh et al. “Deep learning models of the retinal response to natural scenes”. In: *Advances in neural information processing systems* 29 (2016).

[78] Chethan Pandarinath et al. “Inferring single-trial neural population dynamics using sequential auto-encoders”. In: *Nature methods* 15.10 (2018), pp. 805–815.

[79] Guillaume Bellec et al. “Fitting summary statistics of neural data with a differentiable spiking network simulator”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 18552–18563.

[80] Manuel Molano-Mazon et al. “Synthesizing realistic neural population activity patterns using generative adversarial networks”. In: *arXiv preprint arXiv:1803.00338* (2018).

[81] Poornima Ramesh, Mohamad Atayi, and Jakob H Macke. “Adversarial training of neural encoding models on population spike trains”. In: *Real Neurons {\&} Hidden Units: Future directions at the intersection of neuroscience and artificial intelligence@ NeurIPS 2019*. 2019.

[82] Trung Le and Eli Shlizerman. “Stndt: Modeling neural population activity with spatiotemporal transformers”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 17926–17939.

[83] Joel Ye et al. “Neural data transformer 2: multi-context pretraining for neural spiking activity”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 80352–80374.

[84] Mehdi Azabou et al. “A unified, scalable framework for neural population decoding”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 44937–44956.

[85] Jaivardhan Kapoor et al. “Latent diffusion for neural spiking data”. In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 118119–118154.

[86] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[87] Adriano Barenco et al. “Elementary gates for quantum computation”. In: *Physical review A* 52.5 (1995), p. 3457.

[88] Ashley Montanaro. “Quantum algorithms: an overview”. In: *npj Quantum Information* 2.1 (2016), pp. 1–8.

[89] Peter W Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[90] Lov K Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[91] Yasunobu Nakamura, Yu A Pashkin, and JS Tsai. “Coherent control of macroscopic quantum states in a single-Cooper-pair box”. In: *nature* 398.6730 (1999), pp. 786–788.

[92] Juan I Cirac and Peter Zoller. “Quantum computations with cold trapped ions”. In: *Physical review letters* 74.20 (1995), p. 4091.

[93] Chris Monroe et al. “Demonstration of a fundamental quantum logic gate”. In: *Physical review letters* 75.25 (1995), p. 4714.

[94] Daniel Loss and David P DiVincenzo. “Quantum computation with quantum dots”. In: *Physical Review A* 57.1 (1998), p. 120.

[95] Ronald Hanson et al. “Spins in few-electron quantum dots”. In: *Reviews of modern physics* 79.4 (2007), pp. 1217–1265.

[96] Guido Burkard, Daniel Loss, and David P DiVincenzo. “Coupled quantum dots as quantum gates”. In: *Physical Review B* 59.3 (1999), p. 2070.

[97] J. R. Petta et al. “Coherent Manipulation of Coupled Electron Spins in Semiconductor Quantum Dots”. In: *Science* 309.5744 (2005), pp. 2180–2184. DOI: 10.1126/science.1116955. eprint: https://www.science.org/doi/pdf/10.1126/science.1116955. URL: https://www.science.org/doi/abs/10.1126/science.1116955.

[98] Benoit Bertrand et al. “Quantum manipulation of two-electron spin states in isolated double quantum dots”. In: *Physical review letters* 115.9 (2015), p. 096801.

[99] WIL Lawrie et al. “Simultaneous single-qubit driving of semiconductor spin qubits at the fault-tolerant threshold”. In: *Nature Communications* 14.1 (2023), p. 3617.

[100] Francesco Borsoi et al. “Shared control of a 16 semiconductor quantum dot crossbar array”. In: *Nature Nanotechnology* 19.1 (2024), pp. 21–27.

[101] Anasua Chatterjee et al. “Semiconductor qubits in practice”. In: *Nature Reviews Physics* 3.3 (2021), pp. 157–177.

[102] Leo P Kouwenhoven et al. “Electron transport in quantum dots”. In: *Mesoscopic electron transport* (1997), pp. 105–214.

[103] Wilfred G Van der Wiel et al. “Electron transport through double quantum dots”. In: *Reviews of modern physics* 75.1 (2002), p. 1.

[104] JM Elzerman et al. “Single-shot read-out of an individual electron spin in a quantum dot”. In: *nature* 430.6998 (2004), pp. 431–435.

[105] Christian Barthel et al. “Rapid single-shot measurement of a singlet-triplet qubit”. In: *Physical Review Letters* 103.16 (2009), p. 160503.

[106] RJ Schoelkopf et al. “The radio-frequency single-electron transistor (RF-SET): A fast and ultrasensitive electrometer”. In: *science* 280.5367 (1998), pp. 1238–1242.

[107] DJ Reilly et al. “Fast single-charge sensing with a rf quantum point contact”. In: *Applied Physics Letters* 91.16 (2007).

[108] Y-Y Liu et al. “Radio-frequency reflectometry in silicon-based quantum dots”. In: *Physical Review Applied* 16.1 (2021), p. 014057.

[109] Timothy A Baart et al. “Computer-automated tuning of semiconductor double quantum dots into the single-electron regime”. In: *Applied Physics Letters* 108.21 (2016).

[110] CJ Van Diepen et al. “Automated tuning of inter-dot tunnel coupling in double quantum dots”. In: *Applied Physics Letters* 113.3 (2018).

[111] Maxime Lapointe-Major et al. “Algorithm for automated tuning of a quantum dot into the single-electron regime”. In: *Physical Review B* 102.8 (2020), p. 085301.

[112] Lijun Ding and Ardeshir Goshtasby. “On the Canny edge detector”. In: *Pattern recognition* 34.3 (2001), pp. 721–725.

[113] Rui Sun et al. “Survey of image edge detection”. In: *Frontiers in Signal Processing* 2 (2022), p. 826967.

[114] Sandesh S Kalantre et al. “Machine learning techniques for state recognition and auto-tuning in quantum dots”. In: *npj Quantum Information* 5.1 (2019), p. 6.

[115] Justyna P Zwolak et al. “Autotuning of double-dot devices in situ with machine learning”. In: *Physical review applied* 13.3 (2020), p. 034075.

[116] Jana Darulová et al. “Autonomous tuning and charge-state detection of gate-defined quantum dots”. In: *Physical Review Applied* 13.5 (2020), p. 054005.

[117] Joshua Ziegler et al. “Automated extraction of capacitive coupling for quantum dot systems”. In: *Physical Review Applied* 19.5 (2023), p. 054077.

[118] Renato Durrer et al. “Automated tuning of double quantum dots into specific charge states using neural networks”. In: *Physical Review Applied* 13.5 (2020), p. 054019.

[119] Stefanie Czischek et al. “Miniaturizing neural networks for charge state autotuning in quantum dots”. In: *Machine Learning: Science and Technology* 3.1 (2021), p. 015001.

[120] Victor Yon et al. “Experimental Online Quantum Dots Charge Autotuning Using Neural Networks”. In: *Nano Letters* 25.10 (2025), pp. 3717–3725.

[121] Jonas Schuff et al. “Fully autonomous tuning of a spin qubit”. In: *arXiv preprint arXiv:2402.03931* (2024).

[122] Justyna P Zwolak and Jacob M Taylor. “Colloquium: Advances in automation of quantum dot devices control”. In: *Reviews of modern physics* 95.1 (2023), p. 011006.

[123] Justyna P Zwolak et al. “Data needs and challenges for quantum dot devices automation”. In: *npj Quantum Information* 10.1 (2024), p. 105.

[124] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. “An introduction to quantum machine learning”. In: *Contemporary Physics* 56.2 (2015), pp. 172–185.

[125] Maria Schuld and Francesco Petruccione. *Machine learning with quantum computers*. Vol. 676. Springer, 2021.

[126] Amira Abbas et al. “The power of quantum neural networks”. In: *Nature Computational Science* 1.6 (2021), pp. 403–409.

[127] Marcello Benedetti et al. “Parameterized quantum circuits as machine learning models”. In: *Quantum science and technology* 4.4 (2019), p. 043001.

[128] Iris Cong, Soonwon Choi, and Mikhail D Lukin. “Quantum convolutional neural networks”. In: *Nature Physics* 15.12 (2019), pp. 1273–1278.

[129] El Amine Cherrat et al. “Quantum vision transformers”. In: *arXiv preprint arXiv:2209.08167* (2022).

[130] Yanan Li et al. “Quantum recurrent neural networks for sequential learning”. In: *Neural Networks* 166 (2023), pp. 148–161.

[131] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. “A rigorous and robust quantum speed-up in supervised machine learning”. In: *Nature Physics* 17.9 (2021), pp. 1013–1017.

[132] Edward Grant et al. “Hierarchical quantum classifiers”. In: *npj Quantum Information* 4.1 (2018), p. 65.

[133] Koji Terashi et al. “Event classification with quantum machine learning in high-energy physics”. In: *Computing and Software for Big Science* 5 (2021), pp. 1–11.

[134] Marco Cerezo et al. “Challenges and opportunities in quantum machine learning”. In: *Nature computational science* 2.9 (2022), pp. 567–576.

[135] Maria Schuld and Nathan Killoran. “Is quantum advantage the right goal for quantum machine learning?” In: *Prx Quantum* 3.3 (2022), p. 030101.

[136] James Stokes et al. “Quantum natural gradient”. In: *Quantum* 4 (2020), p. 269.

[137] Amira Abbas et al. “On quantum backpropagation, information reuse, and cheating measurement collapse”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 44792–44819.

[138] Amira Abbas et al. “Challenges and opportunities in quantum optimization”. In: *Nature Reviews Physics* (2024), pp. 1–18.

[139] Jarrod R McClean et al. “Barren plateaus in quantum neural network training landscapes”. In: *Nature communications* 9.1 (2018), p. 4812.

[140] Martin Larocca et al. “A review of barren plateaus in variational quantum computing”. In: *arXiv preprint arXiv:2405.00781* (2024).

[141] Jinkai Tian et al. “Recent advances for quantum neural networks in generative learning”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.10 (2023), pp. 12321–12340.

[142] Manuel S Rudolph et al. “Trainability barriers and opportunities in quantum generative modeling”. In: *npj Quantum Information* 10.1 (2024), p. 116.

[143] Alistair Letcher, Stefan Woerner, and Christa Zoufal. “Tight and efficient gradient bounds for parameterized quantum circuits”. In: *Quantum* 8 (2024), p. 1484.

[144] Marcello Benedetti et al. “A generative modeling approach for benchmarking and training shallow quantum circuits”. In: *npj Quantum information* 5.1 (2019), p. 45.

[145] Kaitlin Gili et al. “Do quantum circuit born machines generalize?” In: *Quantum Science and Technology* 8.3 (2023), p. 035021.

[146] Mohammad H Amin et al. “Quantum boltzmann machine”. In: *Physical Review X* 8.2 (2018), p. 021050.

[147] Amir Khoshaman et al. “Quantum variational autoencoder”. In: *Quantum Science and Technology* 4.1 (2018), p. 014001.

[148] Pierre-Luc Dallaire-Demers and Nathan Killoran. “Quantum generative adversarial networks”. In: *Physical Review A* 98.1 (2018), p. 012324.

[149] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. “Quantum generative adversarial networks for learning and loading random distributions”. In: *npj Quantum Information* 5.1 (2019), p. 103.

[150] Haozhen Situ et al. “Quantum generative adversarial network for generating discrete distribution”. In: *Information Sciences* 538 (2020), pp. 193–208.

[151] Elton Yechao Zhu et al. “Generative quantum learning of joint probability distribution functions”. In: *Physical Review Research* 4.4 (2022), p. 043092.

[152] He-Liang Huang et al. “Experimental quantum generative adversarial networks for image generation”. In: *Physical Review Applied* 16.2 (2021), p. 024051.

[153] Shu Lok Tsang et al. “Hybrid quantum–classical generative adversarial network for high-resolution image generation”. In: *IEEE Transactions on Quantum Engineering* 4 (2023), pp. 1–19.

[154] Ira N Levine, Daryle H Busch, and Harrison Shull. *Quantum chemistry*. Vol. 6. Pearson Prentice Hall Upper Saddle River, NJ, 2009.

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

[156] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6 (June 1982), pp. 467–488. ISSN: 1572-9575. DOI: 10.1007/BF02650179. URL: https://doi.org/10.1007/BF02650179.

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

[158] Sandro Sorella. "Wave function optimization in the variational Monte Carlo method". In: *Physical Review B—Condensed Matter and Materials Physics* 71.24 (2005), p. 241103.

[159] Román Orús. "Tensor networks for complex quantum systems". In: *Nature Reviews Physics* 1.9 (2019), pp. 538–550.

[160] J Ignacio Cirac et al. "Matrix product states and projected entangled pair states: Concepts, symmetries, theorems". In: *Reviews of Modern Physics* 93.4 (2021), p. 045003.

[161] David Ceperley and Berni Alder. "Quantum monte carlo". In: *Science* 231.4738 (1986), pp. 555–560.

[162] Julien Toulouse, Roland Assaraf, and Cyrus J Umrigar. "Introduction to the variational and diffusion Monte Carlo methods". In: *Advances in Quantum Chemistry*. Vol. 73. Elsevier, 2016, pp. 285–314.

[163] Midhun T Augustine. "A survey on universal approximation theorems". In: *arXiv preprint arXiv:2407.12895* (2024).

[164] Giuseppe Carleo and Matthias Troyer. "Solving the quantum many-body problem with artificial neural networks". In: *Science* 355.6325 (2017), pp. 602–606.

[165] Matija Medvidović and Javier Robledo Moreno. "Neural-network quantum states for many-body physics". In: *The European Physical Journal Plus* 139.7 (2024), pp. 1–26.

[166] Geoffrey E Hinton. "A practical guide to training restricted Boltzmann machines". In: *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 599–619.

[167] Luciano Loris Viteritti, Francesco Ferrari, and Federico Becca. "Accuracy of restricted Boltzmann machines for the one-dimensional J_1-J_2 Heisenberg model". In: *SciPost Physics* 12.5 (2022), p. 166.

[168] Thomas Spriggs et al. "Quantum resources of quantum and classical variational methods". In: *Machine Learning: Science and Technology* 6.1 (2025), p. 015042.

[169] Robin Thériault, Francesco Tosello, and Daniele Tantari. "Modelling structured data learning with restricted boltzmann machines in the teacher-student setting". In: *Neural Networks* (2025), p. 107542.

[170] Paul Smolensky et al. "Information processing in dynamical systems: Foundations of harmony theory". In: (1986).

[171] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[172] Geoffrey E Hinton. “Deep belief networks”. In: *Scholarpedia* 4.5 (2009), p. 5947.

[173] Adam Coates, Andrew Ng, and Honglak Lee. “An analysis of single-layer networks in unsupervised feature learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 215–223.

[174] Miguel A Carreira-Perpinan and Geoffrey Hinton. “On contrastive divergence learning”. In: *International workshop on artificial intelligence and statistics*. PMLR. 2005, pp. 33–40.

[175] Giuseppe Carleo, Yusuke Nomura, and Masatoshi Imada. “Constructing exact representations of quantum many-body systems with deep neural networks”. In: *Nature communications* 9.1 (2018), p. 5322.

[176] Matija Medvidović and Giuseppe Carleo. “Classical variational simulation of the quantum approximate optimization algorithm”. In: *npj Quantum Information* 7.1 (2021), p. 101.

[177] Kacper Cybiński et al. “Speak so a physicist can understand you! TetrisCNN for detecting phase transitions and order parameters”. In: *arXiv preprint arXiv:2411.02237* (2024).

[178] Felix Frohnert and Evert van Nieuwenburg. “Explainable representation learning of small quantum states”. In: *Machine Learning: Science and Technology* 5.1 (2024), p. 015001.

[179] Anna Dawid et al. “Modern applications of machine learning in quantum sciences”. In: *arXiv preprint arXiv:2204.04198* (2022).

[180] Agnes Valenti et al. “Correlation-enhanced neural networks as interpretable variational quantum states”. In: *Physical Review Research* 4.1 (2022), p. L012010.

[181] Riccardo Rende et al. “Fine-tuning neural network quantum states”. In: *Physical Review Research* 6.4 (2024), p. 043280.

[182] Luciano Loris Viteritti et al. “Transformer wave function for two dimensional frustrated magnets: Emergence of a spin-liquid phase in the Shastry-Sutherland model”. In: *Physical Review B* 111.13 (2025), p. 134411.

3

Exploring Biological Neuronal Correlations with Quantum Generative Models

Understanding of how biological neural networks process information is one of the biggest open scientific questions of our time. Advances in machine learning and artificial neural networks have enabled the modeling of neuronal behavior, but classical models often require a large number of parameters and highly task-specific architectures, which can complicate model design and scalability. Quantum computing offers an alternative approach through quantum machine learning, which can achieve efficient training with fewer parameters. In this work, we introduce a quantum generative model framework for generating synthetic data that captures the spatial and temporal correlations of biological neuronal activity. Our model demonstrates the ability to achieve reliable outcomes with fewer trainable parameters compared to classical methods. These findings highlight the potential of quantum generative models to provide new tools for modeling and understanding neuronal behavior, offering a promising avenue for future research in neuroscience.

The results of this chapter have been published as: **V. Hernandes**, E. Greplova, "Exploring biological neuronal correlations with quantum generative models", Cell Reports Physical Science **6**, 8 (2025) [1]

3.1. INTRODUCTION

Exploring the information processing within biological neuronal networks remains a core challenge in contemporary science, with direct implications across disciplines like neuroscience, medicine, and deep learning [2, 3, 4, 5]. One way to approach this problem is to use computational models that can reproduce the neuronal activity data produced in real systems. Accurate synthetic data can be extremely useful to study properties such as network connectivity and response to stimuli under controlled conditions [6, 7].

Several models for neuronal activity have been developed, and many achieve outstanding results in replicating neuronal network correlations. One class of methods that use statistical mechanics tools to model neuronal activity are Maximum Entropy models, which reliably capture some network correlations by only fitting pairwise interactions [8, 9]. Even though numerous adaptations of this technique have been implemented to achieve higher accuracy or to include temporal correlations [10, 11, 12, 13, 14, 15], this approach shows several limitations when addressing larger networks, especially due to its assumption that pairwise correlations are sufficient to encapsulate most of the statistical features of these complex systems [16, 17].

Another effective approach for modeling neuronal activity is to use machine learning (ML) models to produce data that fits the biological network statistics and use the model to further investigate the properties of the real system. The ML models do not rely on prior information about the biological system but instead learn to reproduce correlations solely from data. A supervised strategy using Convolutional Neural Networks first showed that a deep learning approach can be successful at generating neural responses from stimuli [18]. However, the model's benefits are hampered by limited accuracy and its dependency on labeled data. With the increasing popularization of generative models, models with superior predictive performance and generalization power were implemented. Models like Variational Auto-Encoders [19], Recurrent Neural Networks [20], Generative Adversarial Networks (GANs) [21], and Transformers [22] have been used to produce spike trains (binary sequence representing neuronal activity) with high accuracy and good correspondence of spatial and temporal correlations when compared to real data. While each iteration of these models improves in quality, all share the same disadvantage regarding their interpretability. In order to fit the statistics of larger systems, these models need to use a number of trainable parameters that scales unfavourably with the number of simulated neurons. Apart from demanding more computational power, excessive number of parameters make the models difficult to analyze or be used as a tool to investigate concrete properties of biological networks.

As the field of quantum computing rapidly advances, quantum machine learning (QML) models are rising as an alternative to classical methods, with the possibility of achieving similar results while keeping the parametrized model more compact in terms of trainable parameters [23, 24, 25, 26]. Specifically, the field of quantum generative learning received much attention recently: quantum models have shown better generalization and expressivity for specific tasks when compared to their classical counterparts [27, 28, 29]. Since the conception of QML, one class of quantum generative models has been extensively studied: quantum generative adversarial networks (QGANs) [30]. The adversarial approach has proven successful and is being continually improved, producing higher-dimensional data with more stable training routines [31, 32, 33, 34, 35].

This work is inspired by the observation that quantum generative models have shown promise in replication of discrete distributions [34, 36]. Additionally, the salamander retina dataset has been used as a benchmark for distribution learning using quantum Boltzmann machines [37, 38]. These observations suggest a possibility of full reconstruction of both spatial and temporal correlations with a quantum generative model that we present here. We build on our preliminary work [39] and design SpiQGAN, an efficient quantum framework that enables the production of synthetic neuronal data for biological neuronal networks. SpiQGAN generates spike trains of neuronal activity: data that consist of binary activation states of the neurons obtained from recording the response of ganglion cells of the salamander retina to a visual stimulus as a function of time [40]. This data set represents one of the standard benchmarks in neuronal activation modelling.

To achieve generation of the data that maximally resembles the real biological sample, we apply a hybrid quantum generative adversarial network, with a quantum generator that produces synthetic activity data, and a classical critic that aims to distinguish real data from the dataset [40] from those produced by the quantum generator. The model is trained adversarially, and the outcome is a generator that can reproduce neuronal activity that is to the high degree similar to the salamander retina dataset. Compared to classical neural networks alternatives, the quantum generator has the advantage of achieving reliable outcomes with a significantly reduced number of trainable parameters, that scale more favourably for increasing systems' sizes: the number of parameters is linear in the number of neurons. In other words, SpiQGAN is able to reproduce the behavior of this complex neuronal data set in both space and time with significantly fewer trainable parameters than classical ML models, thus forming a stepping stone towards using quantum approaches for more compact and more interpretable models for neuronal behavior.

3.2. RESULTS

3.2.1. DISTRIBUTIONAL SIMILARITY BETWEEN GENERATED AND REAL DATA

We trained SpiQGAN for $t = \{1, 5, 10, 20, 30\}$ timesteps and for $n = \{2, 4, 6, 8, 10\}$ neurons, in order to evaluate the quality of the generated data as a function of system size and time trace length.

Comparing the distribution of possible states of the simulated data to that of the salamander retina data is a straightforward way to evaluate the quality of our generative model. Reconstruction of these distributions also allows us to calculate distributions distances such as the Jensen-Shannon (JS) divergence. However, direct distribution comparison is particularly challenging: for n neurons and t time steps, the number of possible (spiking) states is 2^{nt}. We are thus able to directly visually compare distributions only for a small number of neurons.

For two neurons, $n = 2$, and and one time step, $t = 1$, the possible states are $\{00, 01, 10, 11\}$. For two neurons and two time steps, $t = 2$, the possible states are $\{0000, 0001, ..., 1110, 1111\}$. In this representation, the first two bits represent neurons 1 and 2 at time step 1 and the last two the states of these neurons at time step 2. This means that the distribution of states quickly becomes intractable for increasing number of neurons or time steps.

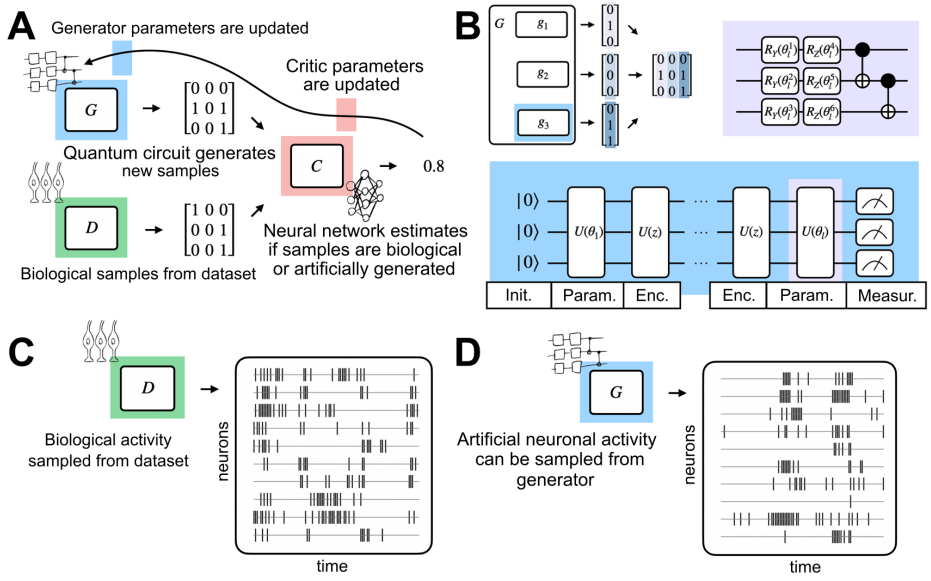


Figure 3.1: Illustration of the model architecture. (A) Architecture of the model, with generator G producing generated samples, and dataset D producing biological samples, which are both used as input for critic C. (B) Architecture of generator. In the upper left corner, the generator composed of several sub-generators is shown. The bottom part shows that each sub-generator is a quantum circuit following a re-uploading scheme. Here a noise-encoding layer and a parametrized layer are repeated for l layers, with the parametrized layer ansatz of each parametrized layer shown in the top right side. After trained, the generator can be used to produce samples (D) similar to samples obtained from the biological dataset (C).

Nonetheless, it is very informative to compare distributions directly for low number of neurons.

For all cases, regardless of system size, we calculated a series of statistical values useful to evaluate the behavior of the generated and the real data from the salamander retina dataset. Specifically, we calculate the pairwise covariance between the activation state of a pair of neurons; the mean firing rate, which correspondent to how many times a neuron spikes per second; the k-probability, equal to the probability of k neurons being active at the same time; and the autocorrelogram to estimate the correlation between a trace of spikes and itself for delayed timesteps.

First we consider the case with a unique timestep (one subgenerator quantum circuit), and neurons varying from 2 to 8. In these cases the distribution is easily numerically tractable. We show the final distribution of generated spiking states, compared to the distribution of the real data in 3.2, with a zoom on the most prominent terms of the distribution in the bottom inset. The probabilities of the spiking states are calculated using the last iteration of the trained circuit. This is coincident with the last value of the JS divergence, which steadily decreases during training, visible in the bottom insets of

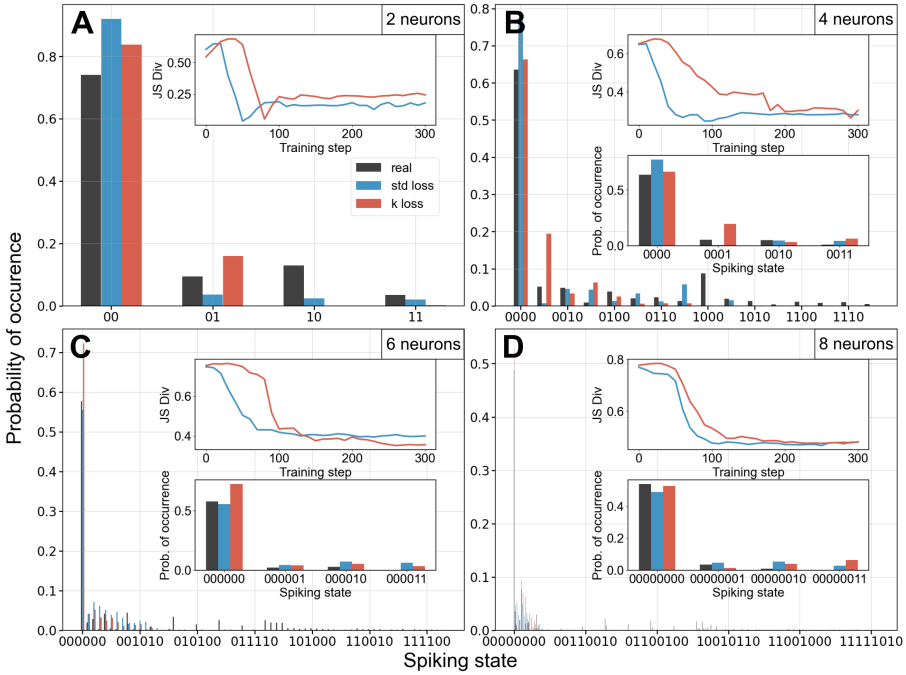


Figure 3.2: Comparison between distribution of states and JS divergence calculated using generated and real data. Each panel show the distribution of spiking states for generated data obtained after training with the K-loss (in red) and with the standard loss (in blue), and the real distribution of the spiking states (black), for (A) 2, (B) 4, (C) 6, and (D) 8 neurons, all for the case of 1 timestep. The bottom inset shows a zoom of the first four activation states. The upper inset shows the JS divergence for all training steps, for K (red) and standard (blue) loss.

3.2. These results show that for a sufficient number of training steps the distribution of generated states converges to the distribution of the salamander retina dataset. This distribution convergence is a first indication that the training is working as intended, and the samples produced by the generator match some of the statistics of the real data. Throughout, we compared both standard loss and biologically inspired K-loss, defined in the experimental procedures section. We found that on average K-loss performed slightly better (see Supplementary Note S1 and Fig. 3.4 for a detailed comparison).

3.2.2. STATISTICAL ANALYSIS OF GENERATED ACTIVITY

In Fig. 3.3 we show further statistics used to assess the quality of the generated data, for 2 and 10 neurons and varying the number of time steps, focusing only on the biologically informed K-loss from now on. Complete results for all neuron numbers and timesteps, accompanied by a focused comparison between the statistics obtained with two different loss functions, are shown in the Supplementary Text (Supplementary Note S1) and figures of the Supplementary Material (see Fig. **??** for statistical metrics and Fig. **??** for activity traces). In Fig. 3.3(H), we see that number of generated spikes as a function of simulated time steps gets closer to the real distribution as the number of time steps

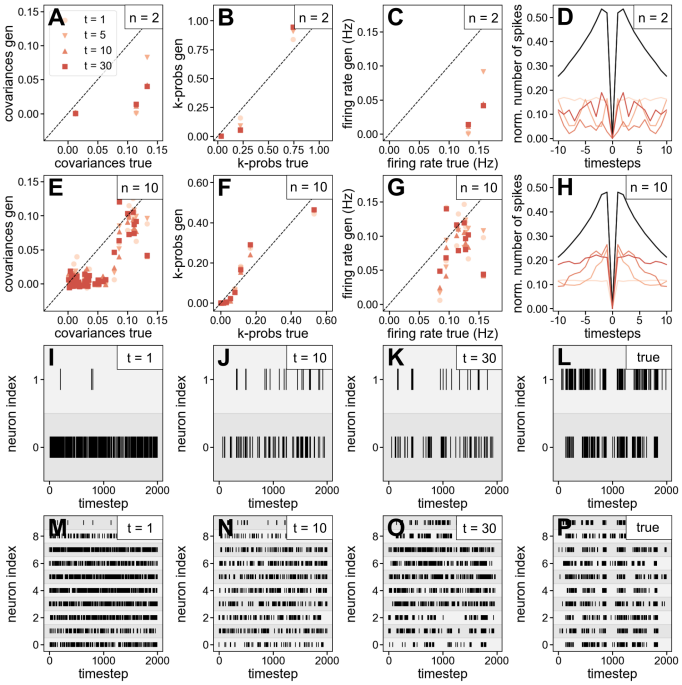


Figure 3.3: Statistics and generated data for 2 and 10 neurons. (A-H) Statistics for the case of 2 and 10 neurons, with 1, 5, 10, and 30 timesteps represented with different colors in each image. Specifically, (A,E) pairwise covariance, (B,F) k-probability, (C,G) firing rate, and (D,H) autocorrelogram are shown. (I-P) Spike traces for 2 and 10 neurons, for the case of generated data with 1 (I,M), 10 (J,N), and 30 timesteps (K,O), and for real data (L,P).

increases.

3.2.3. Spike Train Comparison with Biological Data

Visual comparison of spike trains generated by SpiQGAN and those from the biological dataset is shown in Fig. 3.3 for 2 (I-L) and 10 (M-P) neurons. A visual comparison between the generated spikes and the salamander retina samples shows that for increasing number of time steps the QGAN generated samples start forming bursting clusters, an important feature of the biological dataset.

3.3. Discussion

Overall, all SpiQGAN iterations we implemented achieved a reasonable fit of the key statistical features of the data, especially given the model's simplicity and general purpose design, while maintaining a low number of parameters, which scale favorably (linearly) in the number of neurons. Specifically, for our model, with 4 parameterized layers, the total number of trainable parameters is equal to 8 times the number of neurons per time step. This scaling has the following implication: data generation for neuronal network with dozens of neurons in our implementation uses hundreds of trainable parameters,

compared to thousands, or tens of thousands, in the case of the traditional machine learning approaches [21, 22]. A detailed comparison benchmarking our quantum generator against classical fully-connected generators, which consistently underperformed even with more parameters, can be found in Supplementary Note S2 and Figs. **??**. Moreover, it is clear that models that simulate more neurons presented improved performance, which insinuates that using larger circuits could return even better results. While some metrics like mean firing rate and pairwise covariance show discrepancies, others such as the k-probability and the similarity between generate and real spike train patterns, including burst-like patterns, indicate the model captures essential features of the neuronal data.

We have shown that Quantum Generative Adversarial Networks are able to generate synthetic neuronal activity data that faithfully reproduce both spatial and temporal correlations of the biological dataset. We designed and implemented a resource efficient SpiQGAN that re-uses the same building block across the model. Additionally, we included a biologically informed loss function to take into account statistical properties of the generated samples.

This work lays the foundation for the utilization of quantum learning models beyond quantum science, here in neuroscience modeling. In particular, SpiQGAN opens the possibility of running resource efficient algorithms on quantum computers to beneficially model neuronal activity. With the compact quantum models, the dynamics and interpretation of neuronal activity can be efficiently explored in future work.

3.4. METHODS

3.4.1. GENERATIVE ADVERSARIAL NETWORKS

Generative Adversarial Networks (GANs), introduced by Goodfellow et al. [41], are a powerful class of generative models that learn to synthesize data samples by framing the learning process as an adversarial game between two neural networks: a generator and a discriminator. The generator network, G, aims to produce data samples that mimic those drawn from the true data distribution P_x. It takes a noise vector z, sampled from a predefined distribution P_z (e.g., a Gaussian or uniform distribution), and transforms it into a synthetic data sample, $G(z)$. Meanwhile, the discriminator network, D, acts as a binary classifier, distinguishing between real samples from the true data distribution and fake samples generated by G.

The training objective is formulated as a minimax game, where the generator tries to minimize the probability of the discriminator correctly identifying generated samples, while the discriminator simultaneously maximizes its ability to correctly classify the samples:

$$\min_G \max_D E_{x\sim P_x}\left[\log D(x)\right] + E_{z\sim P_z}\left[\log(1-D(G(z)))\right]. \tag{3.1}$$

Although GANs have achieved remarkable success in various applications (e.g., image synthesis, text generation), they are often plagued by training instabilities such as vanishing gradients and mode collapse [42]. These issues arise primarily because the loss function may not provide meaningful gradients when the discriminator is too strong or too weak, leading to poor convergence.

The Wasserstein GAN (WGAN) [42] addresses many of the training challenges associated with standard GANs by leveraging the Wasserstein distance (also known as the Earth-Mover distance) to measure the divergence between the true data distribution and the generated data distribution. Unlike the original GAN, the discriminator in WGAN, referred to as a critic, outputs a scalar value instead of a binary classification, quantifying how well the generated samples approximate the real data distribution. The WGAN objective is formulated as:

$$\min_G \max_{D \in \mathscr{D}} E_{x \sim P_x}[D(x)] - E_{z \sim P_z}[D(G(z))], \tag{3.2}$$

where $\mathscr{D}$ is the set of all 1-Lipschitz functions, enforced through weight clipping or gradient penalties [43]. By stabilizing the gradients, WGAN significantly improves convergence behavior, allowing the generator to learn a more accurate representation of the target distribution.

3.4.2. PARAMETRIZED QUANTUM CIRCUITS AND QUANTUM GANS

As quantum computing has advanced, quantum machine learning has emerged as a promising frontier. One key concept is Parametrized Quantum Circuits (PQCs). PQCs consist of a sequence of quantum gates with parameters that are classically optimized. PQCs can encode complex quantum states and can be used to approximate complex distributions.

Building on this foundation, Quantum Generative Adversarial Networks (QGANs) extend the GAN framework into the quantum domain by incorporating quantum components such as quantum generators, quantum discriminators, or both. In QGANs, the generator may be implemented as a PQC, which is trained to generate samples that match the desired distribution.

3.4.3. IMPLEMENTATION OF THE QUANTUM GENERATOR AND THE CLASSICAL CRITIC

SpiQGAN uses a quantum generator to model the spike activity patterns of retinal ganglion cells. Specifically, we employ a Patch WQGAN approach, where the quantum generator is divided into several sub-generators, each corresponding to a different timestep. Each sub-generator shares the same PQC architecture but has independent trainable parameters, allowing for flexibility in capturing the temporal dynamics of neuronal activity.

The generator begins with a random initial quantum state $|z\rangle$, which is mapped to the final state $|g\rangle$ using a data re-uploading scheme [44]. The quantum circuit consists of five layers, where each layer applies a sequence of parametrized unitaries $U(\theta_i)$ and noise-encoding unitaries $U(z)$. The parametrized unitary $U(\theta_i)$ is implemented using rotation gates around the Y and Z axes (R_Y and R_Z) and entangling operations (CNOT gates) between adjacent qubits, while the encoding block applies R_X rotations to each qubit to encode a sampled noise vector. The generator outputs a sequence of activity states for multiple neurons over several timesteps by concatenating the outputs from all sub-generators. All quantum circuit results in this work are obtained from classical numerical simulations of ideal quantum hardware. This allows us to benchmark the behavior of the model without the influence of noise. This comes with the cost of longer

training times, given the inefficienty of calcuting quantum gradient. The range of training times in this range from approximately 2 hours for the smallest system, to 5 days for the larger systems. We also tested generators with a higher number of variational layers per sub-generator to evaluate whether deeper circuits would improve performance. However, the improvements were not significant, while training became significantly more expensive due to the cost of quantum gradient estimation. For this reason, we fixed the number of layers to five in all reported experiments, balancing expressivity and trainability.

The critic in our QGAN framework is a fully connected classical neural network. The network consists of:

- An input layer matching the size of the generated samples,
- A hidden layer with 64 neurons using ReLU activation,
- An output layer without an activation function, which directly provides a scalar value representing the divergence between the real and generated distributions.

3.4.4. TRAINING PROCEDURE

SpiQGAN is trained by optimizing two separate loss functions for the generator and the critic. The critic's loss function aims to maximize the difference between its outputs for real samples x and generated samples $G(z)$:

$$L_C = \frac{1}{2B}\sum_j \big(C(G(z_j)) - C(x_j)\big), \tag{3.3}$$

whereas the generator's objective is to minimize the critic's evaluation of the generated samples:

$$L_G = -\frac{1}{B}\sum_j C(G(\boldsymbol{z_j})) - K\left(\sum_i G(z_j)^i - x_j^i\right), \tag{3.4}$$

with B being the batch size, and the term $K(\sum_i G(z_j)^i - x_j^i)$ added to the standard Wasserstein's generator loss function, inspired by Maximum-Entropy models [12], corresponding to the difference between the number of spikes in a fake sample and those in a real sample. This loss was named K-loss, and by setting $K = 0$ the standard loss is retrieved. Training alternates between two updates of the critic and one update of the generator, ensuring stable convergence. The Adam optimizer is employed with learning rates of 0.05 for the generator and 0.002 for the critic.

3.4.5. DATASET AND EVALUATION METRICS

The dataset is comprised of neuronal spike activity recorded from retinal ganglion cells in a salamander retina [40]. It contains 297 repetitions of a 19-second natural movie, recorded as binary spike events, where 1 indicates a spike and a 0 indicates no spike. The goal is to generate synthetic data that replicates these binary spike patterns while maintaining important statistical properties. To evaluate the performance of the QGAN, we used the following statistical metrics:

- **Pairwise Covariance:** Measures the extent to which two neurons fire together. High covariance suggests that the neurons are more likely to spike simultaneously.
- **Mean Firing Rate:** The average rate at which a neuron fires spikes over time. This metric helps ensure that the generated data matches the overall activity level of the real data.
- **k-Probability:** The probability distribution over the number of spikes (k) in a given time window. Matching this distribution ensures that the generated data captures the variability in spike counts.
- **Autocorrelogram:** A measure of the temporal structure of the spike train, representing the correlation of a neuron's spike times with itself over different time lags. This metric is crucial for capturing the temporal dynamics of neuronal activity.

To comprehensively assess the model's performance, we conducted experiments varying the number of neurons n={2,4,6,8,10} and timesteps t={1,2,5,10,20,30 }. For small-scale systems, alongside the metrics listed above, we computed the exact probabilities of all possible spiking states, used to compare the generated and real data distributions using distance measures such as Jensen-Shannon divergence, which, for two probability distributions P and Q is defined as:

$$\mathrm{JS}(P \parallel Q) = \frac{1}{2}\mathrm{KL}(P \parallel M) + \frac{1}{2}\mathrm{KL}(Q \parallel M), \tag{3.5}$$

where $M = \frac{1}{2}(P+Q)$, and KL denotes the Kullback-Leibler divergence:

$$\mathrm{KL}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}. \tag{3.6}$$

3.5. SUPPLEMENTARY MATERIAL

3.5.1. SUPPLEMENTAL NOTES

NOTE S1

In Fig. 3.4 we show a comparison between the biological K-loss and standard loss. For all combinations of (neurons, timesteps, loss function), we calculate the mean square error between statistics (k-probability and firing rate) obtained with SpiQGAN generated and real samples. Figs. 3.4(A) and (D) show the error for both losses for all timesteps as a function of the number of neurons. Figs. 3.4(B) and (E) show the error for all neurons as a function of the number of timesteps. The error visibly decreases for increasing number of neurons. Interestingly, the same is not true for increasing number of timesteps, suggesting that even the time correlations (see Fig. 3.3(H)) are better fitted by using more qubits rather than by increased number of parametrized circuits in the generator. Figs. 3.4(C) and (D) shows the difference between the mean-square error obtained using the K-loss and the standard loss, for all combinations of (neurons, timesteps). A positive value mean that the error using the K-loss is lower, while a negative value means that the error obtained with the standard loss is lower. We see that for most cases, the values are positive, indicating the K-loss achieves a better fit for the majority of models, especially for the k-probability (panel C).

NOTE S2

To benchmark the performance of our quantum generator, we compared it with existing classical models applied to the same dataset. Notably, classical models generally require a significantly larger number of parameters and must be tailored specifically to the patterns in neuronal data. To establish a fair baseline for comparison with our quantum generator, we implemented a classical fully-connected generator and varied its hyperparameters to create several classical models with different parameter counts. Despite extensive hyperparameter searches, none of the classical training runs converged well, suggesting that our quantum generator provides a more flexible and general approach, while classical networks require highly task-specific design.

We evaluated the classical GAN in two configurations: 2 neurons and 2 timesteps, and 6 neurons and 10 timesteps. For both settings, we varied learning rates, training iterations, and the size of the generator's hidden layers, resulting in over 80 hyperparameter combinations and a wide range of parameter counts.

In the smaller configuration (neurons = 2, timesteps = 2), the classical generator consistently failed to capture meaningful structure. Autocorrelogram analysis revealed a lack of temporal correlation, and the generated samples often exhibited mode collapse, producing nearly identical outputs. Increasing the number of parameters did not significantly improve performance. We show selected results for this case in Fig. 3.5.

A similar trend was observed in the larger configuration (neurons = 6, timesteps = 10), with representative results shown in Fig. 3.6. While the autocorrelogram showed minor signs of improvement, indicating weak temporal structure in a few cases, the overall quality of the generated data remained poor. Importantly, the classical models tested in this case already exceeded the number of parameters used in our quantum model (which has fewer than 1000 parameters), further supporting the claim that the quantum generator achieves better fidelity with fewer parameters.

3.5.2. SUPPLEMENTAL FIGURES

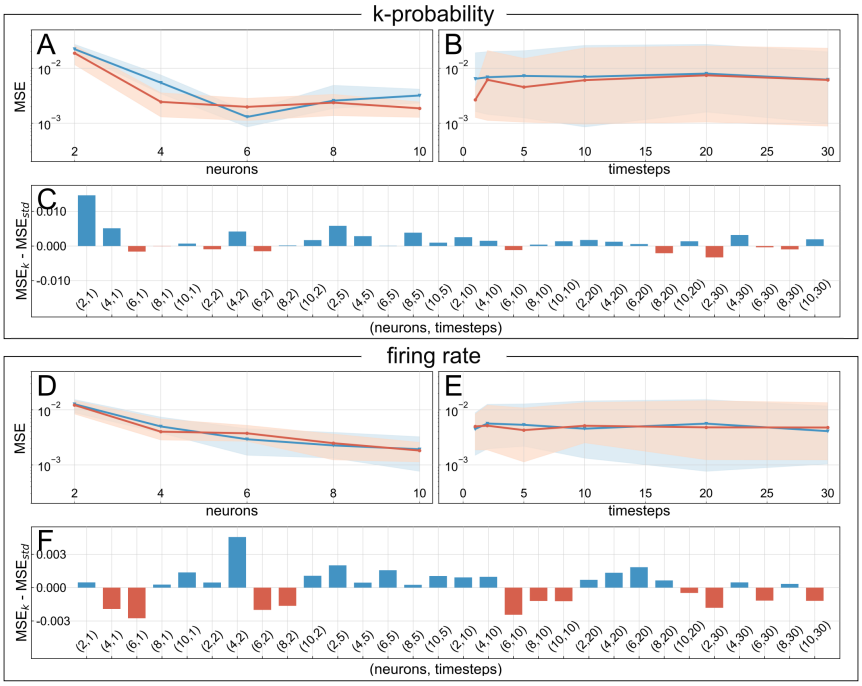


Figure 3.4: Mean-Square Error of k-probability and firing for models using K-loss and standard loss. (A) Error of k-probability value for K (orange) and standard (blue) loss as a function of the number of neurons. (B) Error of k-probability value for K and standard loss as a function of the number of timesteps. (C) Difference between the mean-square error of k-probability obtained using the K-loss and the standard loss, for all combinations of (neurons, timesteps). (D) Error of firing rate value for K (orange) and standard (blue) loss as a function of the number of neurons. (E) Error of firing rate value for K and standard loss as a function of the number of timesteps. (F) Difference between the mean-square error of firing rate obtained using the K-loss and the standard loss, for all combinations of (neurons, timesteps). Data are represented as mean (solid lines) +/- standard deviation (shaded area).

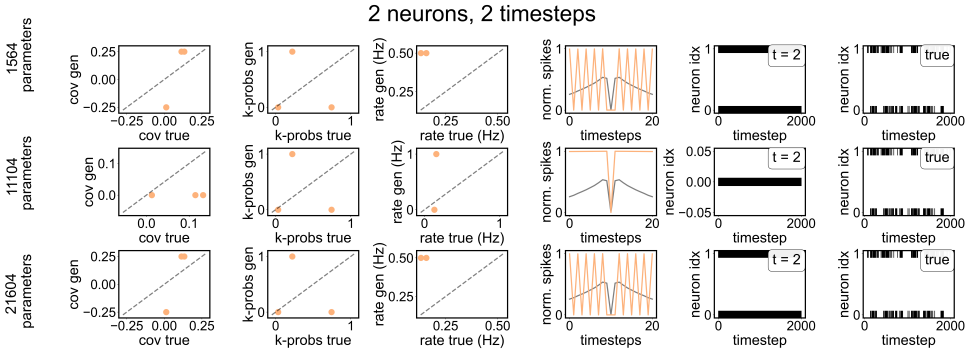


Figure 3.5: Statistics and generated data for 2 neurons and 2 timesteps, using classical GAN. From left to right, columns show pairwise covariance, k-probability, firing rate, autocorrelogram, spike traces for the case of generated data, and spike traces for the case of real data. Each row shows the result for a models with different number of parameters in the generator.

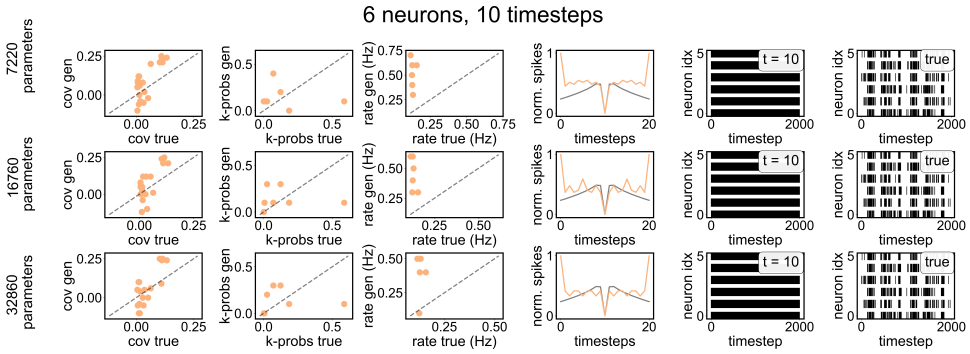


Figure 3.6: Statistics and generated data for 6 neurons and 10 timesteps, using classical GAN. Same as 3.5, for 6 neurons and 10 timesteps.

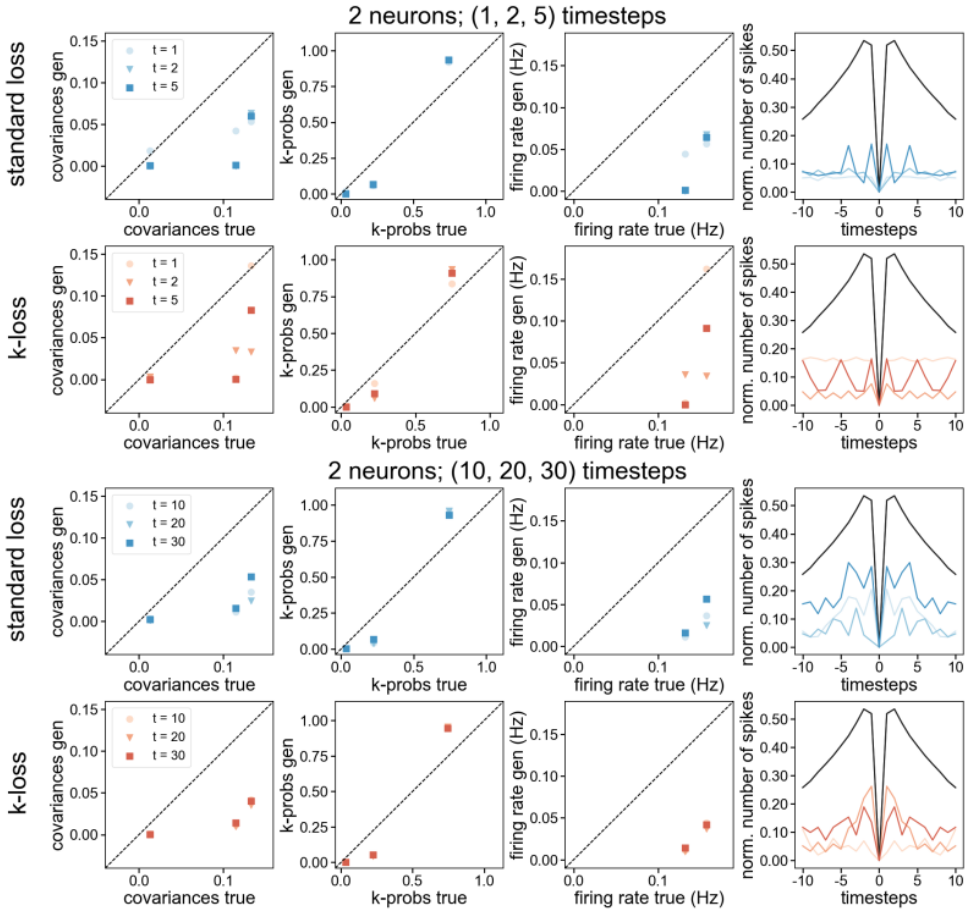


Figure 3.7: Statistics for 2 neurons. From left to right: pairwise covariance, k-probability, firing rate, and auto-correlogram. The first row shows the results for the model that uses standard loss, for the case of 1, 2, and 5 timesteps. The second row shows the results for the model that uses K-loss, for the case of 1, 2, and 5 timesteps. Third and fourth rows show the results for 10, 20, and 30 timesteps, for models using standard loss (third) and K-loss (fourth).

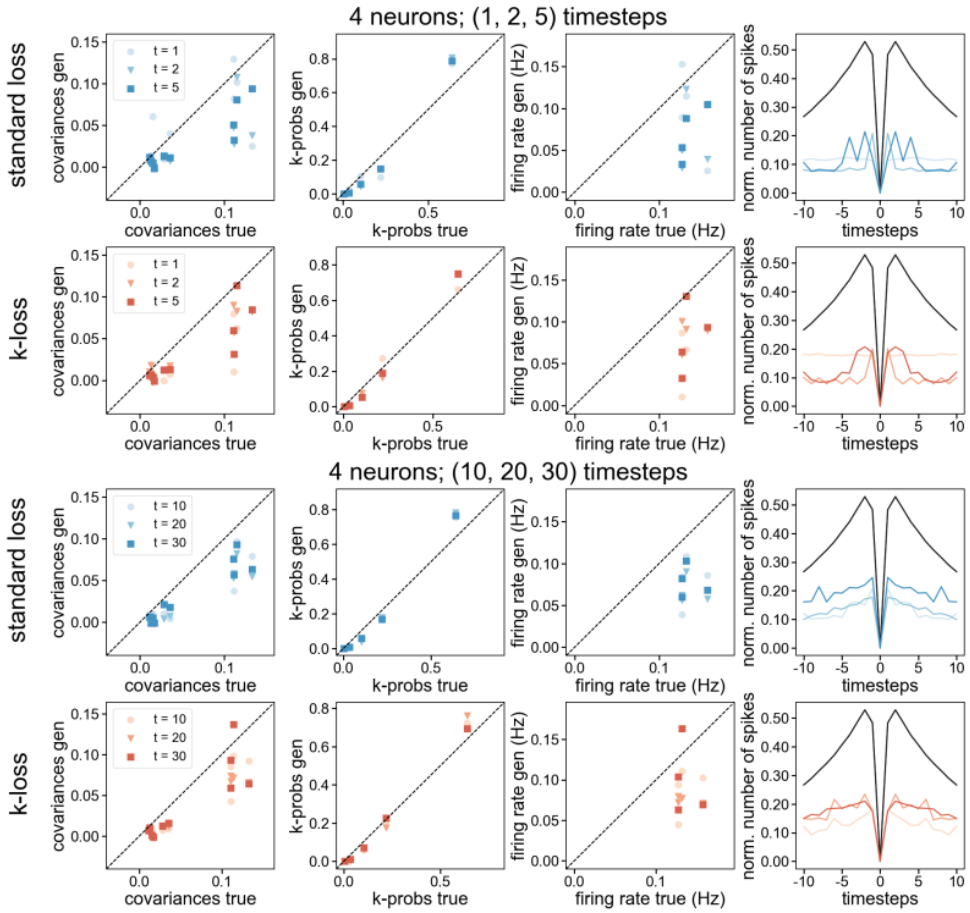


Figure 3.8: Statistics for 4 neurons. Same as 3.7, for 4 neurons.

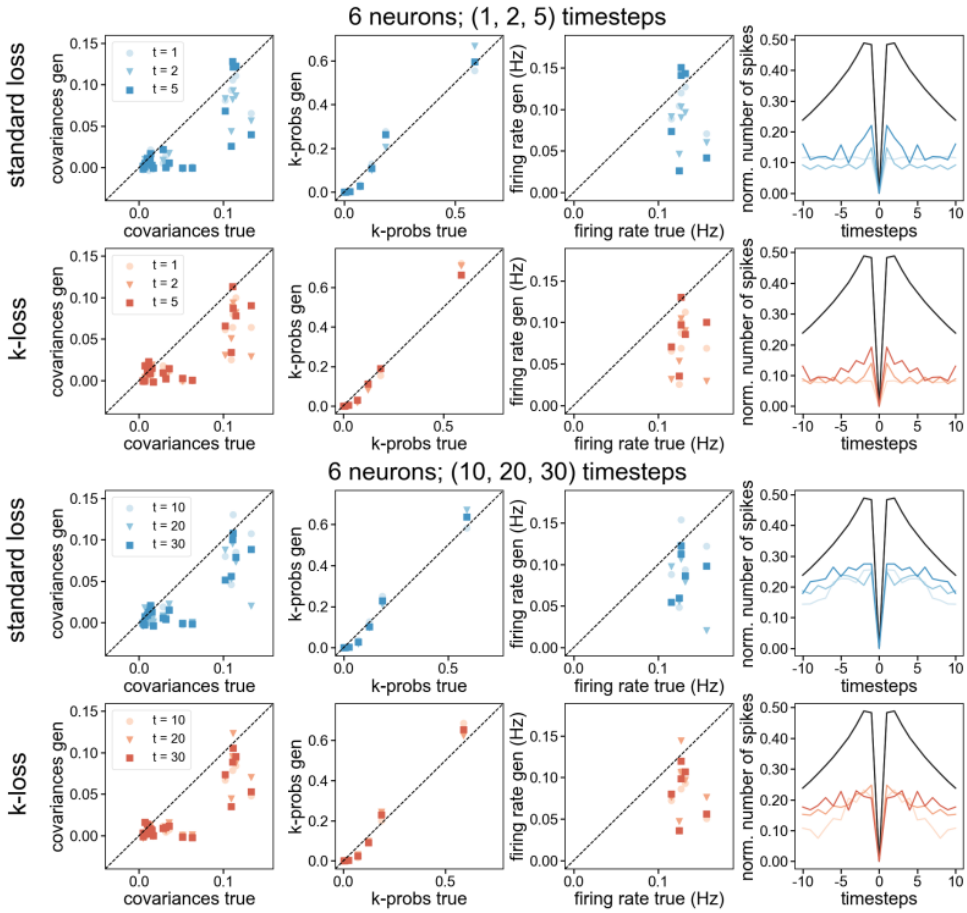


Figure 3.9: Statistics for 6 neurons. Same as 3.7, for 6 neurons.

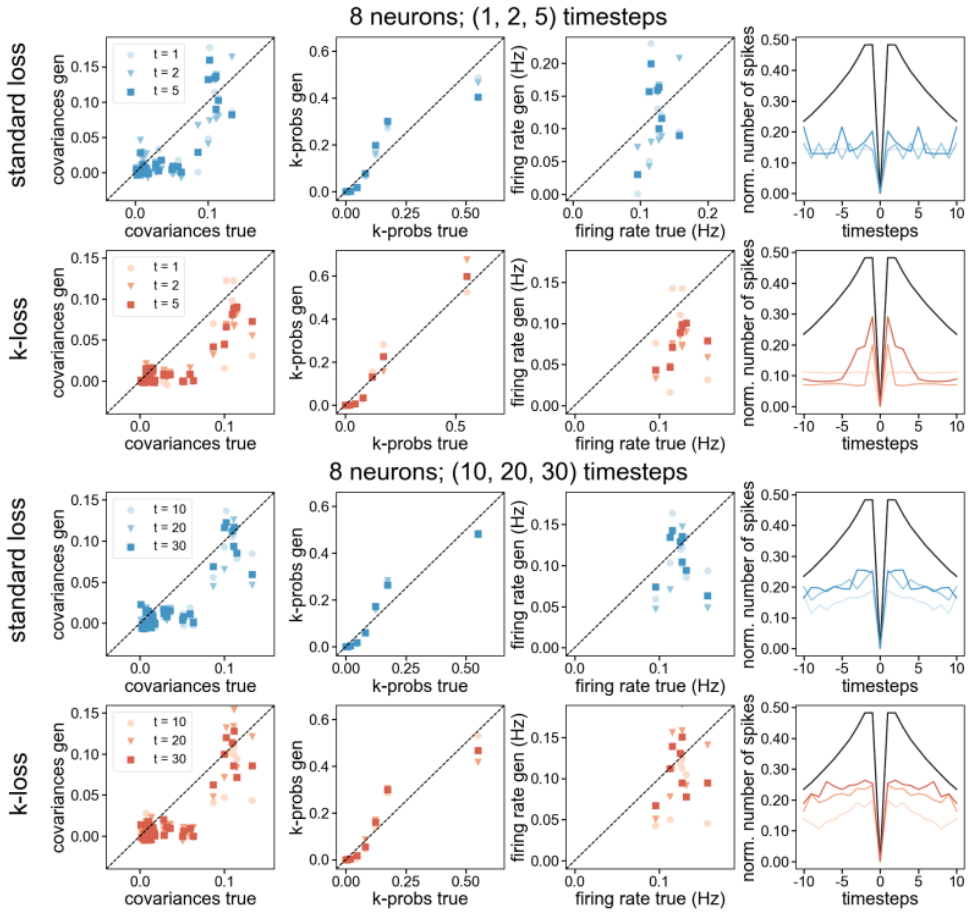


Figure 3.10: Statistics for 8 neurons. Same as 3.7, for 8 neurons.

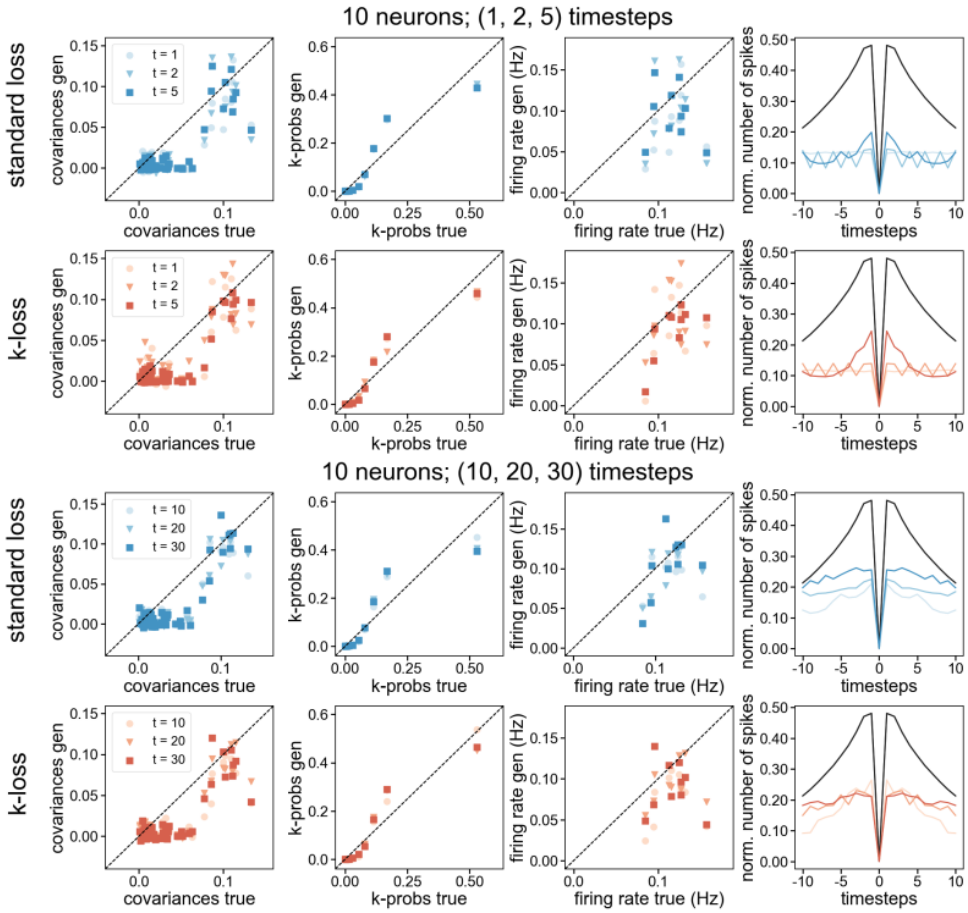


Figure 3.11: Statistics for 10 neurons. Same as 3.7, for 10 neurons.

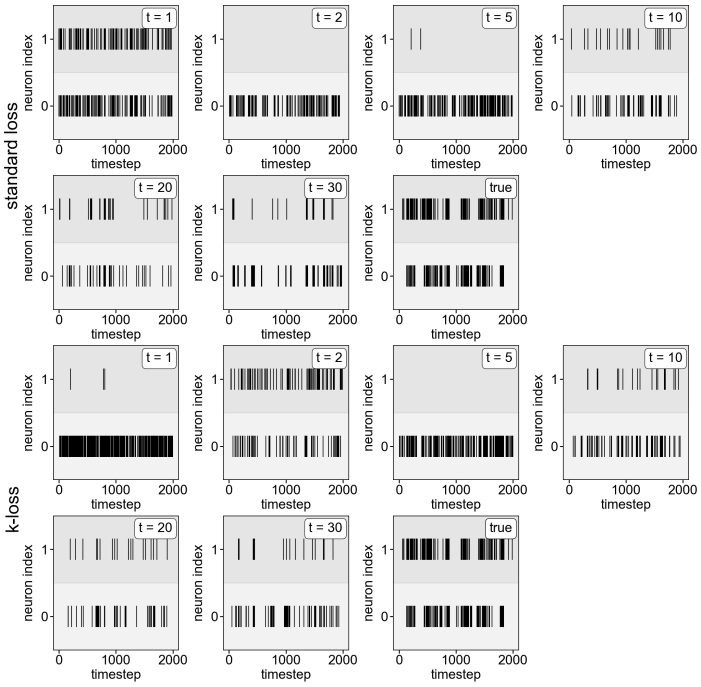


Figure 3.12: Comparison between generated and real data samples for 2 neurons. The first and second rows show spike traces for generated data from trained models with 1, 2, 5, 10, 20, and 30 timesteps, using standard loss, and a spike trace for real dataset. Third and fourth rows show the same as the first two, for the models trained with K-loss.

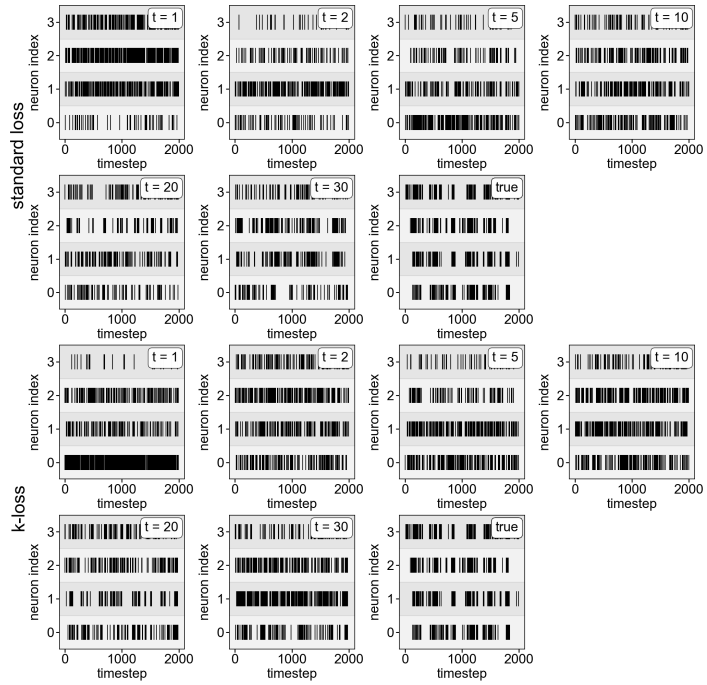


Figure 3.13: Comparison between generated and real data samples for 4 neurons. Same as 3.12, for 4 neurons.

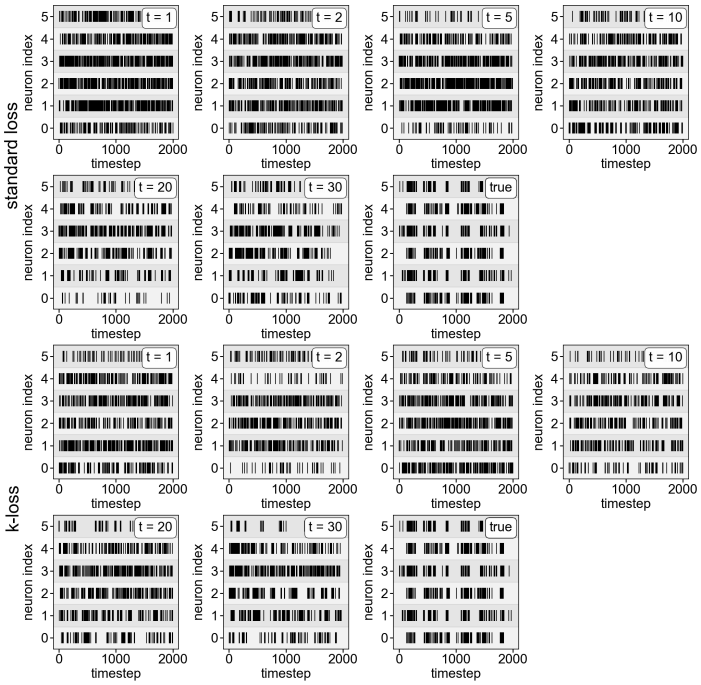


Figure 3.14: Comparison between generated and real data samples for 6 neurons. Same as 3.12, for 6 neurons.

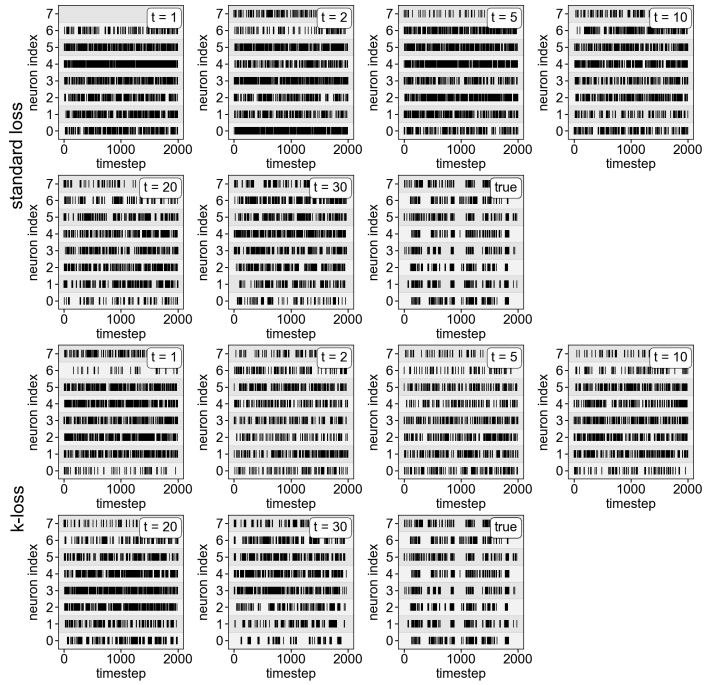


Figure 3.15: Comparison between generated and real data samples for 8 neurons. Same as 3.12, for 8 neurons.

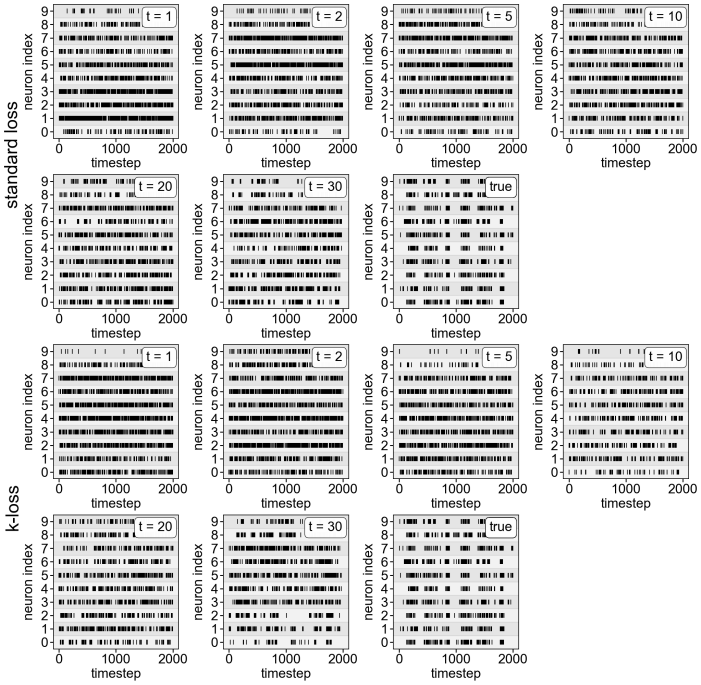


Figure 3.16: Comparison between generated and real data samples for 10 neurons. Same as 3.12, for 10 neurons.

REFERENCES

[1] Vinicius Hernandes and Eliska Greplova. “Exploring biological neuronal correlations with quantum generative models”. In: *Cell Reports Physical Science* 6.8 (2025).

[2] Fred Rieke et al. *Spikes: exploring the neural code*. 1999.

[3] Andrew White et al. “EEG spike activity precedes epilepsy after kainate-induced status epilepticus”. In: *Epilepsia* 51.3 (2010), pp. 371–383.

[4] Demis Hassabis et al. “Neuroscience-inspired artificial intelligence”. In: *Neuron* 95.2 (2017), pp. 245–258.

[5] Andrew Saxe, Stephanie Nelli, and Christopher Summerfield. “If deep learning is the answer, what is the question?” In: *Nature Reviews Neuroscience* 22.1 (2021), pp. 55–67.

[6] Jianbin Wen, Michael Peitz, and Oliver Brüstle. “A defined human-specific platform for modeling neuronal network stimulation in vitro and in silico”. In: *Journal of Neuroscience Methods* 373 (2022), p. 109562.

[7] Johanna Senk et al. “Connectivity concepts in neuronal network modeling”. In: *PLoS Computational Biology* 18.9 (2022), e1010086.

[8] Gasper Tkacik et al. “Ising models for networks of real neurons”. In: *arXiv preprint q-bio/0611072* (2006).

[9] Elad Schneidman et al. “Weak pairwise correlations imply strongly correlated network states in a neural population”. In: *Nature* 440.7087 (Apr. 2006), pp. 1007–1012. ISSN: 1476-4687. DOI: 10.1038/nature04701. URL: https://doi.org/10.1038/nature04701.

[10] Aonan Tang et al. “A Maximum Entropy Model Applied to Spatial and Temporal Correlations from Cortical Networks In Vitro”. In: *Journal of Neuroscience* 28.2 (2008), pp. 505–518. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.3359-07.2008. eprint: https://www.jneurosci.org/content/28/2/505.full.pdf. URL: https://www.jneurosci.org/content/28/2/505.

[11] Olivier Marre et al. “Prediction of spatiotemporal patterns of neural activity from pairwise correlations”. In: *Physical review letters* 102.13 (2009), p. 138101.

[12] Gašper Tkačik et al. “The simplest maximum entropy model for collective behavior in a neural network”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2013.03 (2013), P03011.

[13] Einat Granot-Atedgi et al. “Stimulus-dependent maximum entropy models of neural population codes”. In: *PLoS computational biology* 9.3 (2013), e1002922.

[14] Gašper Tkačik et al. “Searching for collective behavior in a large network of sensory neurons”. In: *PLoS computational biology* 10.1 (2014), e1003408.

[15] Geoffroy Delamare and Ulisse Ferrari. “Time-dependent maximum entropy model for populations of retinal ganglion cells”. In: *Physical Sciences Forum*. Vol. 5. 1. MDPI. 2022, p. 31.

[16] Yasser Roudi, Sheila Nirenberg, and Peter E Latham. “Pairwise maximum entropy models for studying large biological systems: when they can work and when they can’t”. In: *PLoS computational biology* 5.5 (2009), e1000380.

[17] Urs Köster et al. “Modeling higher-order correlations within cortical microcolumns”. In: *PLoS computational biology* 10.7 (2014), e1003684.

[18] Lane T. McIntosh et al. “Deep Learning Models of the Retinal Response to Natural Scenes”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS’16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 1369–1377. ISBN: 9781510838819.

[19] Chethan Pandarinath et al. “Inferring single-trial neural population dynamics using sequential auto-encoders”. In: *Nature methods* 15.10 (2018), pp. 805–815.

[20] Guillaume Bellec et al. “Fitting summary statistics of neural data with a differentiable spiking network simulator”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021. URL: https://openreview.net/forum?id=9DEAT9pDiN.

[21] Manuel Molano-Mazon et al. “Synthesizing realistic neural population activity patterns using Generative Adversarial Networks”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=r1VVsebAZ.

[22] Trung Le and Eli Shlizerman. “STNDT: Modeling Neural Population Activity with Spatiotemporal Transformers”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=iUOUnyS6uTf.

[23] Maria Schuld and Francesco Petruccione. “Learning with Quantum Models”. In: *Supervised Learning with Quantum Computers*. Cham: Springer International Publishing, 2018, pp. 247–272. ISBN: 978-3-319-96424-9. DOI: 10.1007/978-3-319-96424-9_8. URL: https://doi.org/10.1007/978-3-319-96424-9_8.

[24] Maria Schuld et al. “Circuit-centric quantum classifiers”. In: *Physical Review A* 101.3 (2020), p. 032308.

[25] Vedran Dunjko and Peter Wittek. “A non-review of quantum machine learning: trends and explorations”. In: *Quantum Views* 4 (2020), p. 32.

[26] Amira Abbas et al. “The power of quantum neural networks”. In: *Nature Computational Science* 1.6 (2021), pp. 403–409.

[27] Marcello Benedetti et al. “A generative modeling approach for benchmarking and training shallow quantum circuits”. In: *npj Quantum Information* 5.1 (May 27, 2019). Number: 1 Publisher: Nature Publishing Group, pp. 1–9. ISSN: 2056-6387. DOI: 10.1038/s41534-019-0157-8. URL: https://www.nature.com/articles/s41534-019-0157-8 (visited on 07/29/2022).

[28] Yuxuan Du et al. “Expressive power of parametrized quantum circuits”. In: *Physical Review Research* 2.3 (2020), p. 033125.

[29] Yuxuan Du et al. “Power of quantum generative learning”. In: *arXiv preprint arXiv:2205.04730* (2022).

[30] Pierre-Luc Dallaire-Demers and Nathan Killoran. "Quantum generative adversarial networks". In: *Phys. Rev. A* 98 (1 July 2018), p. 012324. DOI: 10.1103/PhysRevA.98.012324. URL: https://link.aps.org/doi/10.1103/PhysRevA.98.012324.

[31] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. "Quantum Generative Adversarial Networks for learning and loading random distributions". In: *npj Quantum Information* 5.1 (Nov. 2019), p. 103. ISSN: 2056-6387. DOI: 10.1038/s41534-019-0223-2. URL: https://doi.org/10.1038/s41534-019-0223-2.

[32] Haozhen Situ et al. "Quantum generative adversarial network for generating discrete distribution". In: *Information Sciences* 538 (2020), pp. 193–208. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2020.05.127. URL: https://www.sciencedirect.com/science/article/pii/S0020025520305545.

[33] He-Liang Huang et al. "Experimental quantum generative adversarial networks for image generation". In: *Physical Review Applied* 16.2 (2021), p. 024051.

[34] Shu Lok Tsang et al. "Hybrid Quantum-Classical Generative Adversarial Network for High Resolution Image Generation". In: *IEEE Transactions on Quantum Engineering* (2023), pp. 1–19. DOI: 10.1109/TQE.2023.3319319.

[35] Nan-Run Zhou et al. "Hybrid quantum–classical generative adversarial networks for image generation via learning discrete distribution". In: *Signal Processing: Image Communication* 110 (2023), p. 116891.

[36] Smit Chaudhary et al. "Towards a scalable discrete quantum generative adversarial neural network". In: *Quantum Science and Technology* 8.3 (2023), p. 035002.

[37] Hilbert J Kappen. "Learning quantum models from quantum or classical data". In: *Journal of Physics A: Mathematical and Theoretical* 53.21 (2020), p. 214001.

[38] Onno Huijgen et al. "Training quantum Boltzmann machines with the β-variational quantum eigensolver". In: *Machine Learning: Science and Technology* 5.2 (2024), p. 025017.

[39] Vinicius Hernandes and Eliska Greplova. "Modeling Neuronal Activity with Quantum Generative Adversarial Networks". In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 2. IEEE. 2023, pp. 330–331.

[40] Olivier Marre et al. "Multi-electrode array recording from salamander retinal ganglion cells". In: (2017).

[41] Ian J Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[42] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks". In: *International conference on machine learning*. PMLR. 2017, pp. 214–223.

[43] Ishaan Gulrajani et al. "Improved training of wasserstein gans". In: *Advances in neural information processing systems* 30 (2017).

[44] Adrián Pérez-Salinas et al. "Data re-uploading for a universal quantum classifier". In: *Quantum* 4 (2020), p. 226.

4

autoMEA: Machine learning-based burst detection for multi-electrode array datasets

Neuronal activity in the highly organized networks of the central nervous system is the vital basis for various functional processes, such as perception, motor control, and cognition. Understanding interneuronal connectivity and how activity is regulated in the neuronal circuits is crucial for interpreting how the brain works. Multi-electrode arrays (MEAs) are particularly useful for studying the dynamics of neuronal network activity and their development as they allow for real-time, high-throughput measurements of neural activity. At present, the key challenge in the utilization of MEA data is the sheer complexity of the measured datasets. Available software offers semi-automated analysis for a fixed set of parameters that allow for the definition of spikes, bursts and network bursts. However, this analysis remains time-consuming, user-biased, and limited by pre-defined parameters. Here, we present autoMEA, software for machine learning-based automated burst detection in MEA datasets. We exemplify autoMEA efficacy on neuronal network activity of primary hippocampal neurons from wild-type mice monitored using 24-well multi-well MEA plates. To validate and benchmark the software, we showcase its application using wild-type neuronal networks and two different neuronal networks modeling neurodevelopmental disorders to assess network phenotype detection. Detection of network char-

The results of this chapter have been published as: *__V. Hernandes__, *A.M. Heuvelmans, V. Gualtieri, D.H. Meijer, †G.M. van Woerden, †E. Greplova, "autoMEA: Machine learning-based burst detection for multi-electrode array datasets", Frontiers in Neuroscience **18**, 1446578 (2024) [1]
*,†These authors contributed equally.

acteristics typically reported in literature, such as synchronicity and rhythmicity, could be accurately detected compared to manual analysis using the autoMEA software. Additionally, autoMEA could detect reverberations, a more complex burst dynamic present in hippocampal cultures. Furthermore, autoMEA burst detection was sufficiently sensitive to detect changes in the synchronicity and rhythmicity of networks modeling neurodevelopmental disorders as well as detecting changes in their network burst dynamics. Thus, we show that autoMEA reliably analyses neural networks measured with the multi-well MEA setup with the precision and accuracy compared to that of a human expert.

4.1. INTRODUCTION

In the human brain, highly orchestrated activity of neuronal networks lie at the basis of various functional neurological processes. In these networks, excitability is tightly regulated through a complex interplay between glutamatergic, excitatory neurons, and GABAergic, inhibitory neurons [2]. In the search to better understand processes contributing to a balanced network, multi-electrode arrays (MEAs) have provided a valuable tool to study the activity of neuronal networks as a whole [3, 4]. MEA devices allow for non-invasive measurement of electrical activity in neuronal cultures *in vitro* [5, 6]. Importantly, it allows one to follow the development of network activity as the culture matures and record responses of the network to compounds of interest [7, 8]. MEA has furthermore proven to be a valuable tool to model neurological diseases *in vitro* [9]. Many neurological diseases have been studied using neuronal networks derived from rodent brain tissue, such as Alzheimer's disease, epilepsy, and various neurodevelopmental disorders (NDDs) [9]. Since researchers are able to develop human induced pluripotent stem cell models of neurological diseases through differentiation into neuronal networks, MEAs have gained even more interest.

MEA electrodes record fluctuations in the electric field around them. When neurons on top of an electrode fire action potentials, the fast flux of sodium and potassium ions across the membrane generates a typical change in the extracellular potential surrounding the MEA electrode, which is classified as a spike [9, 6, 10]. In typical MEA analysis software, spikes can be detected using a threshold for a minimal amplitude deviation from baseline noise (typically +/- 5 standard deviations) [6]. While frequencies of spiking activity can give information about the excitability of a network, this parameter is prone to fluctuations by technical and batch-to-batch variation [11]. More interesting and robust outcome measures are parameters that describe the activity of the network as a whole. During the initial phase of network development *in vitro*, spikes can be detected mostly in a random sequence. However, as the network starts to mature, periods of high-frequency spike trains are interrupted by periods of quiescence [10]. These high-frequency spike trains are classified as bursts. Typically, these bursts are recorded synchronously by multiple electrodes across the culture indicating the formation of a functionally connected network, hence, these are called network bursts (NB). Many parameters can be extracted from this type of activity, such as the synchronicity of the network, the rhythmicity of network activity, and characteristics such as network burst duration and composition.

One major challenge faced when analyzing MEA data is the definition of bursts, about which no consensus has been reached in the research field [9]. Generally, bursts are defined based on a MaxInterval method, which defines a maximum inter-spike interval that is used as a threshold to classify a sequence of spikes as a burst. More extensive methods also include a maximum interval between bursts, the minimum duration of a burst, and a minimum number of spikes fired within a burst [12]. These thresholds can be chosen by an experimenter, or determined using adaptive burst detection algorithms [7, 13]. Electrophysiological mechanisms underlying burst dynamics depend on multiple characteristics, such as neuronal excitability, synaptic transmission, and network connectivity. Hence, burst dynamics may differ between different types of neuronal cultures and change throughout network development [14]. For example, a study by Charlesworth and colleagues (2015) identified a unique feature of hippocampal neuronal cultures when compared to cortical cultures. From 11 days *in vitro* (DIV), hippocampal bursting dynamics were characterized by a theta rhythm, in which a single burst can be divided into multiple reverberations, i.e. short sequences of high-frequency spiking activity that closely follow each other, clustering into a burst [14]. While these reverberations can be detected using the same MaxInterval method, there is a higher chance of interference by spiking noise, e.g. single spikes occurring in between two reverberations thereby merging them together, because the inter-spike intervals will remain below the threshold. It is thus challenging to define a single set of parameters that can reliably define bursts over different experiments and culture types, and more complex burst dynamics may require more adaptive detection methods.

Currently, analysis is often carried out in software provided with the hardware (e.g. Multiwell Screen by Multi Channel Systems). In this software, parameters are set by the experimenter, based on visual inspection of the data, searching for the most ideal parameters for a certain dataset or by thresholding using the log inter-spike interval (ISI) [12, 7, 13]. However, this default set of parameters is often error-prone, especially when the data contains more complex bursting dynamics such as the reverberations in hippocampal cultures. For example, reverberations can also be detected using the MaxInterval method, but may be merged due to a single spike fired in between two reverberations. Current detection methods can only ignore such spiking noise if settings are manually altered by the experimenter through visual inspection. The visual inspection of the data to find the ideal set of parameters and adjust within a parameter's defined range if necessary, is a very labor-intensive process, requiring file-by-file analysis of the data, and creates a risk for experimenter bias and reduces the objectivity of the analysis method. Therefore, often a default set is chosen taking for granted that multiple reverberations may be merged together in more noisy recordings.

Over the past years, several analysis packages for MEA data have been published [15, 16, 17, 18]. While these packages provide more extensive and automated analysis options than software provided with the recording system, the MaxInterval and logISI burst detection methods integrated into this software generally use parameters that are not suitable for the detection of reverberations within bursts (e.g. interburst interval of 100ms). These methods are thus limited to simpler bursting dynamics, and not that of

for example hippocampal bursts. Furthermore, the MEA technique is currently well-adopted in the iPSC field as a relevant technique for functional phenotyping of NDDs [19, 11]. Interestingly, recent studies have started to identify reverberations in NDD models of iPSC-derived neurons. For example, reverberating bursts emerged in iPSC-derived neuronal cultures of Rett syndrome [20], Kabuki syndrome [21], Dravet syndrome [22] and Kleefstra syndrome patients [23]. This further implicates the usefulness of a detection model that can accurately detect these more complex burst dynamics for the broad MEA community.

Machine learning (ML) models have become common in various domains, demonstrating remarkable efficacy and facilitating practical applications in everyday life. In scientific tasks, ML has spread through nearly every field, offering a valuable tool, particularly for tasks requiring automation, such as fine-tuning intricate devices [24, 25] or analyzing complex datasets [26, 27]. In connection to the problem of burst detection described above, ML emerges as a promising solution. Particularly, in scenarios employing the MaxInterval method, human experts must iteratively select parameters and inspect data quality until convergence to an optimal parameter set is achieved. One can exploit the optimal parameter determination process by selecting a set of MEA-data and correspondent optimal MaxInterval parameters and use it to effectively train a ML model to replicate the decision-making of human experts in parameter selection. Here, we have developed an automated analysis software tool, including optimized burst detection using a machine learning approach. We generated a sophisticated noise-resilient algorithm that takes a MEA signal or a spike train as input, and outputs the MaxInterval parameters that return reverberating bursts that would have been manually detected by a human expert. This process is fully automated and does not need any manual assistance by a human operator. We present our algorithmic solution to the burst determination challenge and provide its implementation as a ready-to-use open-source package, autoMEA [28, 29], that the neuronal network community can immediately use and expand upon. We demonstrate that our approach works for a range of different input data (raw measurements averaged in different ways as well as binary spikes data) and we validate the model using existing datasets of two neurodevelopmental disorders (NDDs) across different time points in neuronal network development.

4.2. RESULTS

4.2.1. MACHINE LEARNING MODELS

The autoMEA software detects bursts using two different methods: 1) the default method, with which detection is done using the same MaxInterval parameters as in the manual analysis software, and 2) burst detection based on MaxInterval parameters predicted by a machine learning model. In this work 3 different models were generated and implemented in the software. The three models are all built upon 1D-convolutional artificial neural networks, and have a different architecture depending on the input data: spikes30 model uses a 5-second binary spike trace averaged by 30 time-steps; signal30 uses real-valued signal averaged by 30 time-steps; and signal100 the real-valued signal averaged by 100 time-steps. Schematic depiction of our workflow is shown in Figure 4.1. The de-

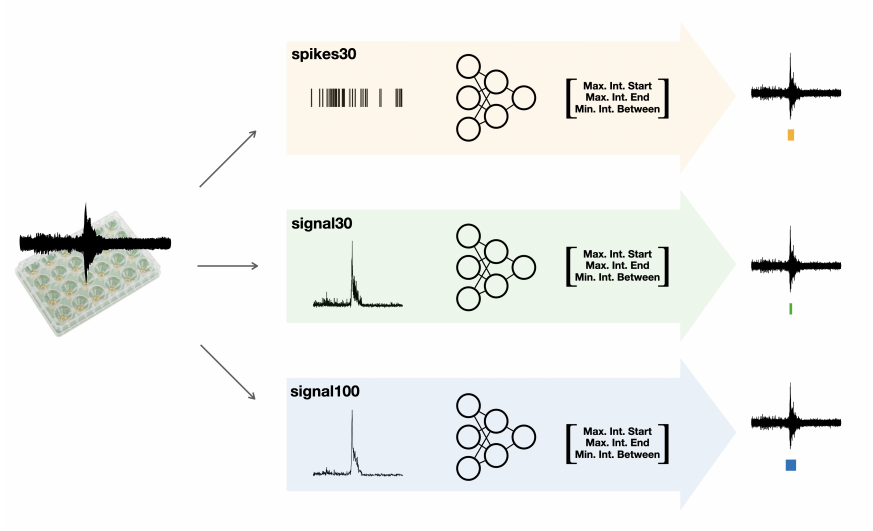


Figure 4.1: Schematic depiction of our workflow. After collecting the raw MEA data, we feed them into three different workflows: we post-process the data either into the form of spikes or averaged signal (over 30 or 100 time bins). We follow by training the neural network specific to each of the inputs (these models are referred to as spikes30, signal30, and signal100) to output key parameters for the MaxInterval method: maximum interval to start and end the reverberations and minimal time between the reverberations. These parameters predicted by each machine learning model are then used for MaxInterval method that predicts the reverberations that are combined into bursts

.

tailed information on the architecture and training of neural network machine learning models spikes30, signal30, and signal 100 is available in the Supplemental Information. The averaging of the original 5-second data was performed to reduce input size, thereby significantly reducing the computational power needed by the models. We tested different averaging window sizes to make sure the model's performance was not compromised. All models' output consist of a three-dimensional array, corresponding to 3 out of the 5 MaxInterval parameters. These predicted parameters are then applied in the standard MaxInterval method to detect reverberating bursts and network activity, of which an example is presented in Figure 4.2.

To train the machine learning models, a relatively small dataset comprising 797 burst samples was utilized. The dataset was built by using a functionality of our package that plots windows containing signal, spikes, reverberations and bursts detected using specific MaxInterval parameters. This feature was used to perform a standard post-processing analysis, where optimal MaxInterval parameters for detecting bursts were selected by the experimenter, and windows of 5-second duration containing signal or spikes or bursts and their corresponding parameters were saved for each sample.

The training of all three types of artificial neural network models (spikes30, signal30, and signal100) employed Mean Squared Error (MSE) as a loss function to measure the distance between the optimal human-selected parameters and those predicted by the network. A thorough hyperparameter tuning process was conducted by experimenting

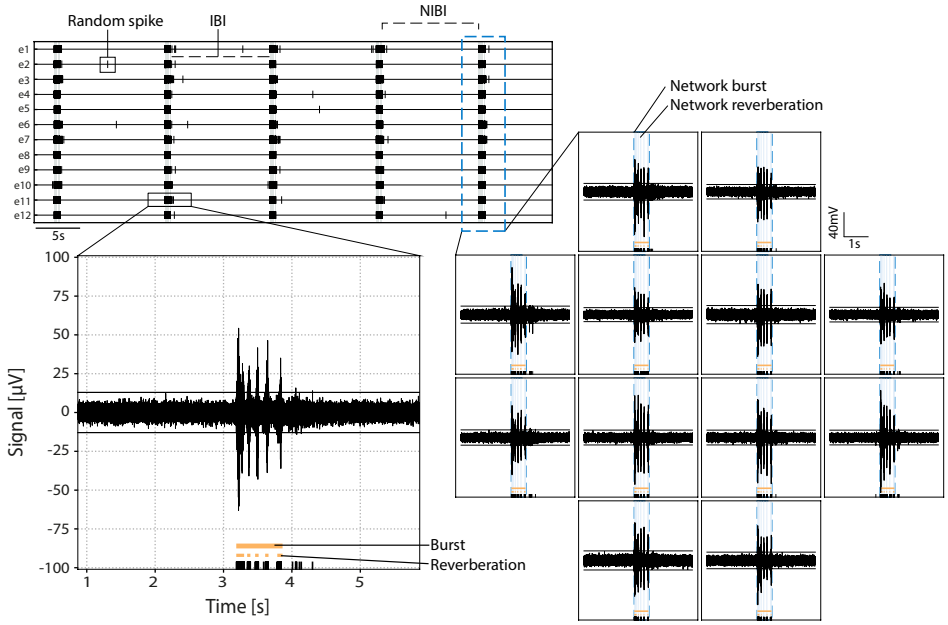


Figure 4.2: Example of activity in primary hippocampal cultures. Top left: rasterplot with activity recorded on 12 electrodes with spikes indicated in black and network reverberations indicated with grey shading. Parameters random spike, bursts and network bursts, IBI (inter-burst interval) and NIBI (network inter-burst interval) are indicated with boxes/lines in the raster. Bottom left: zoom in of the raw signal of a burst in one channel with spikes indicated in black, and reverberations and burst indicated in orange at the bottom. Right: raw signal of activity during a network burst. Bursts and reverberations detected on single channels are indicated in orange, overlaying light blue shade indicates detected network reverberations and the surrounding blue dotted line indicates the detected network burst.

with different layers sizes and activation functions, followed by selecting the best overall hyperparameters for each model. To evaluate the efficacy of the predicted parameters in producing optimal bursts, simply relying on the loss function showing differences between sets of MaxInterval parameters was insufficient since there are multiple MaxInterval method parameter combinations that yield low loss and none of these combinations are captured by a single label consisting of experimenter's parameter choice. Hence, a custom accuracy metric was introduced: it compares the bursts obtained obtained from parameters chosen by the artificial neural network and those selected by the experimenter. The learning curves showing the custom accuracy, for the training and validation set, are shown in Figure 4.3. From the learning curves, we observed that the spikes30 model gradually learned to predict *MaxInterval* parameters throughout the training process. The custom accuracy stayed very close to zero for the first 15 to 25 epochs, and then increased until converging to a value close to 0.86 (Figure 4.3A). In contrast, the signal30 and signal100 models, which use normalized signal as input, achieved a high value of custom accuracy (approximately 0.86) already in the first learning epoch,

and the learning process was mostly visible by a shortening of the shaded area (statistical variation of network's predictions), signaling that the accuracy of the signal30 and signal100 models was converging to a common value (Figure 4.3B-C).

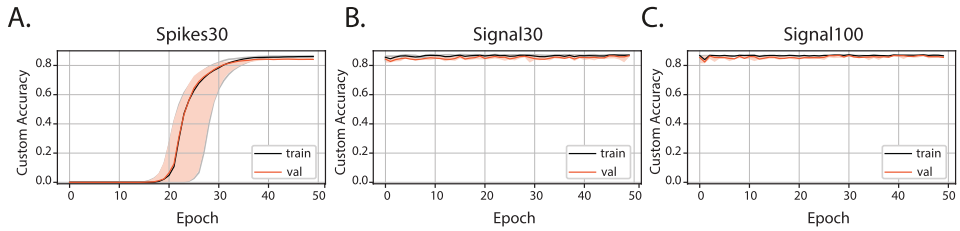


Figure 4.3: Custom accuracy for three machine learning models: spikes30, signal 30, and signal100. The solid line is the median of all custom accuracy values calculated for each epoch, and the shaded area is the range between the minimum and maximum values of custom accuracy for each epoch.

Assessment of burst detection quality

Next, we assessed the accuracy of burst detection by the machine learning model in comparison to using the default MaxInterval parameter. We compared the default parameters and machine learning model as follows: an experienced experimenter was presented with one burst, the detection of its reverberations was presented in two different ways: using the default parameters, and the machine learning model's predicted parameters. Being blind to the detection type, the experimenter then scored the detection as equal or gave a preference for one over the other detected burst. The quantitative overview of this comparison is shown in Figure 4.4. The experimenter scored 120 bursts for each of the three neural network models: spikes30, signal30, and signal100. The majority of bursts were detected equally well by the default method and either of the model's detection as judged by the experimenter. For the spikes30 model, there was an equal amount of bursts that were better detected by the default or machine learning methods. For signal30 and signal100 models,bursts detected using the machine-learning-based parameter prediction were slightly more often the preferred detection. Overall, the detection accuracy was not statistically different between the 3 different models ($\chi^2(4) = 5.814$, p=0.2135).

4.2.2. VALIDATION OF PARAMETER DETECTION

Accuracy of spike and network dynamics detection by the autoMEA software.

Network bursts are typical electrical activity patterns characterized by high-frequency spiking activity happening simultaneously across multiple electrodes in the well. Similar to the MCS Software, in the autoMEA software, spikes are used to detect bursting activity in the network. Thus, for the model to accurately detect network bursts, it was first of all important that the spikes could be correctly detected with the reproduction of the signal and threshold settings in autoMEA software. To this extent, we correlated the MFR of all

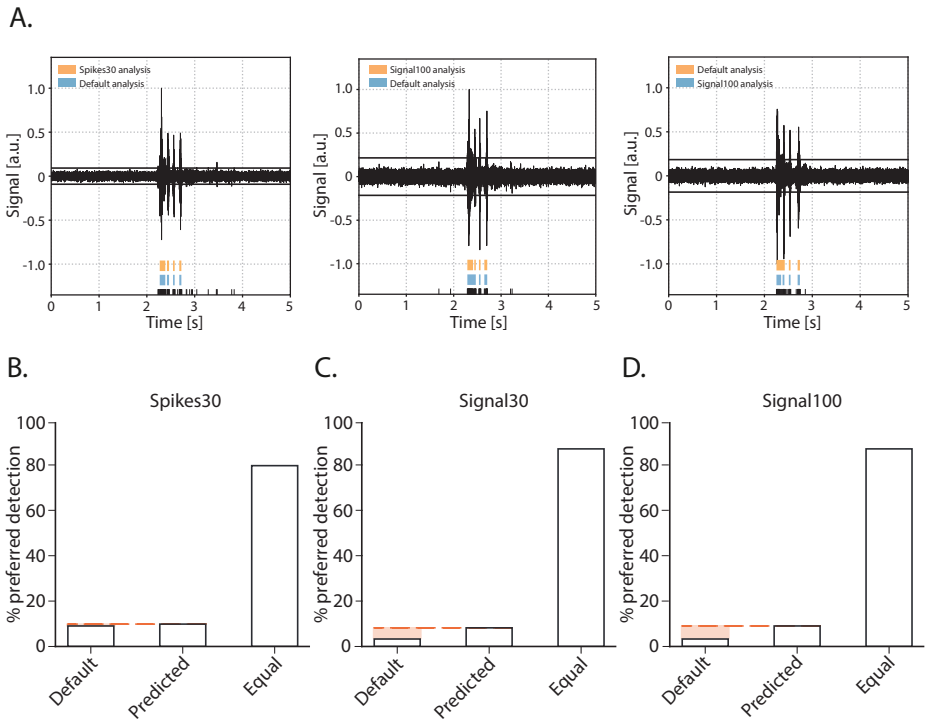


Figure 4.4: Burst quality metric for the parameter prediction models. A) 3 examples of bursts as presented for scoring to an experimenter. Raw signal of a burst with detected spikes indicated in black at the bottom and above the reverberations as detected by either the default method or one of the detection models (spikes30, signal30, signal100). During scoring, the experimenter was blind to which color represented the default and spikes30/signal30/signal100 detection, and color could switch with each presentation of a new burst. e.g. A (left) default in blue, spikes30 in orange, (middle) default in blue, signal100 in orange, (right) default in orange, signal100 in blue. B-D) Burst quality score with the % of bursts scored as preferred with default method, with a predicted model, or equal for each method.

wells in our hippocampal datasets, detected by manual analysis using the MSC software, to the MFR detected by the autoMEA software. We found a near-perfect correlation between the MFR detected by the manual analysis and autoMEA software ($r(79) = 0.9939$, $p < 0.0001$, Figure 4.5).

Subsequently, we assessed the correlation between the manual analysis and the different detection methods: using default parameters, and either of the machine learning prediction models, for a set of outcome measures that can be used to describe neuronal network dynamics (Figure 4.6). We focused here on outcome parameters related to network bursting activity as these have been reported to be more robust than single channel bursting activity, the latter being more sensitive to e.g. technical and batch-to-batch variation [11].
Firstly, the % of random spikes (%RS, i.e. spikes not being part of a network burst) and

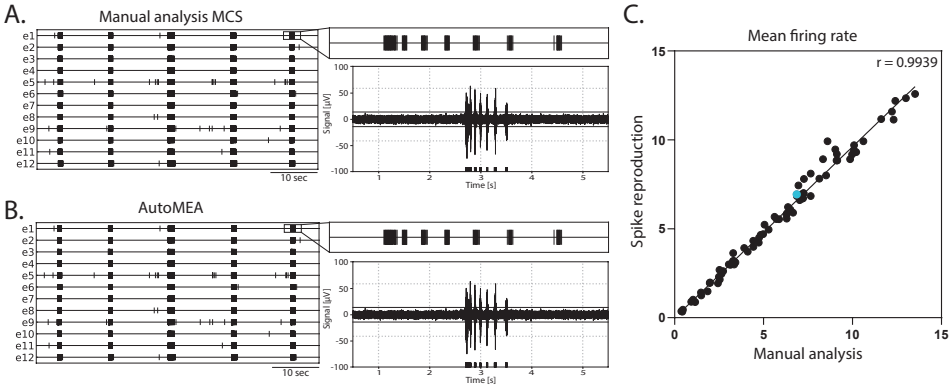


Figure 4.5: Correlation of the mean firing rate between spikes detected with manual analysis in MCS and autoMEA software. A) example raster plot (left) of spikes detected by MCS analysis with a zoom-in of the raster for a 5-second section above a 5-second section of the raw data (right). B) example raster plot (left) of spikes detected by the autoMEA software with a zoom-in of the raster for a 5-second section above a 5-second section of the raw data. C) Mean firing rate in Hz, detected by manual analysis in MCS software on the x-axis, and autoMEA software on the y-axis. The correlation between the two detection methods is near-perfect. The blue dot is the datapoint presented in figures A and B. N = 81 wells

the network burst rate (NBR) can together describe the level of synchronicity in the network (Figure 4.6B1-2). We found that the detection of these parameters by any of the autoMEA models strongly correlated with the manual analysis. For both parameters, all autoMEA models showed correlations of r > 0.9 (statistics are presented in Table 4.1). All autoMEA models slightly overestimated the %RS, while on average fewer network bursts were detected (Figure 4.6B1-2).

Table 4.1: Correlation statistics between manual and autoMEA output for all analyzed parameters. %RS = % random spikes, NBR = network burst rate, NIBI = network inter-burst interval, CoV of NIBI = Coefficient of variance of NIBI, NBD = network burst duration, NBC = network burst composition

Method	%RS	NBR	NIBI	CoV of NIBI	NBD	NBC	Network reverberation duration
Default	0.9315	0.9720	0.9279	0.9239	0.9470	0.9911	0.9882
spikes30	0.9331	0.9284	0.8864	0.9653	0.9446	0.0955	0.9842
signal30	0.9333	0.9586	0.9228	0.9706	0.9499	0.9912	0.9880
signal100	0.9340	0.9623	0.9184	0.9668	0.9512	0.9912	0.9867

Secondly, the rhythmicity of network burst firing can be described by the network inter-burst intervals (NIBI) and more specifically, the coefficient of variance (CoV-NIBI) thereof. These two parameters, determined by all autoMEA models, also strongly correlated with the results of the manual analysis (Figure 4.6C1-2, statistics are presented in Table 4.1). Interestingly, the outcome parameter NIBI showed a difference in the direc-

tionality of the change for the different models. The spikes30 model detected a slightly increased NIBI and showed the least strong correlation to the manual analysis ($r(74) = 0.8864$, $p>0.0001$). The default, signal30 and signal100 results showed strong correlations with manual analysis ($r > 0.9$). For these approaches, the detection resulted in a reduced NIBI compared to the manual analysis.
Finally, network bursts can be characterized by their duration and reverberations. We again correlated the outcome of each autoMEA model to the manual analysis and found a strong correlation between the Network burst duration (NBD), Network burst composition (NBC i.e. network reverberations / network burst), and networkreverberation duration (Figure 4.6D1-3, statistics are presented in Table 4.1). Here, both network reverberation duration and NBC were lower when detected using any autoMEA model, while NBD was slightly higher.

Importantly, we observed that the difference between the prediction models and the manual analysis was mostly driven by the data processing method, as we observed that the default method already introduced small differences in burst detection compared to the manual analysis. The deviation between the default method and the prediction models was very small, showing the autoMEA models accurately reproduced the detection of reverberating network bursts compared to detection by a default parameter set. Only for the outcome parameter NIBI did the spikes30 model deviate more from the default method and the signal prediction models.

Taken together, these results show that we can accurately detect reverberating network bursts using the autoMEA software, in a way that is at least as good as a default set chosen by an experimenter through extensive visual inspection.

4.2.3. VALIDATION OF PHENOTYPE DETECTION

Detection of neuronal network phenotypes in genetic models of NDDs.
To validate the sensitivity of the autoMEA software for phenotype detection in disease models, we compared the analysis of the autoMEA software with manual analysis performed by an experienced researcher. Two different MEA datasets of NDDs were used for this comparison: 1) The RHEB-p.P37L model [30, 31], representing a disorder associated with severe refractory epilepsy due to hyperactivity of the mTOR pathway. The RHEB-p.P37L model has been well characterized on the MEA and showed increased spike and network bursting activity, premature synchronization of network activity, and loss of the reverberating burst pattern [32]. 2) The CAMK2g-p.R292P model [33, 34] for a neurodevelopmental disorder associated with severe intellectual disability, autism, and general developmental delay. This model has not previously been characterized using multi-electrode arrays.

RHEB-p.P37L
In this validation experiment, we used a dataset of neuronal network activity recordings of the RHEB-p.P37L model at two recording days (days *in vitro*: DIV): DIV7 and DIV14. The hippocampal cultures were transduced at DIV1 with a virus inducing the expression

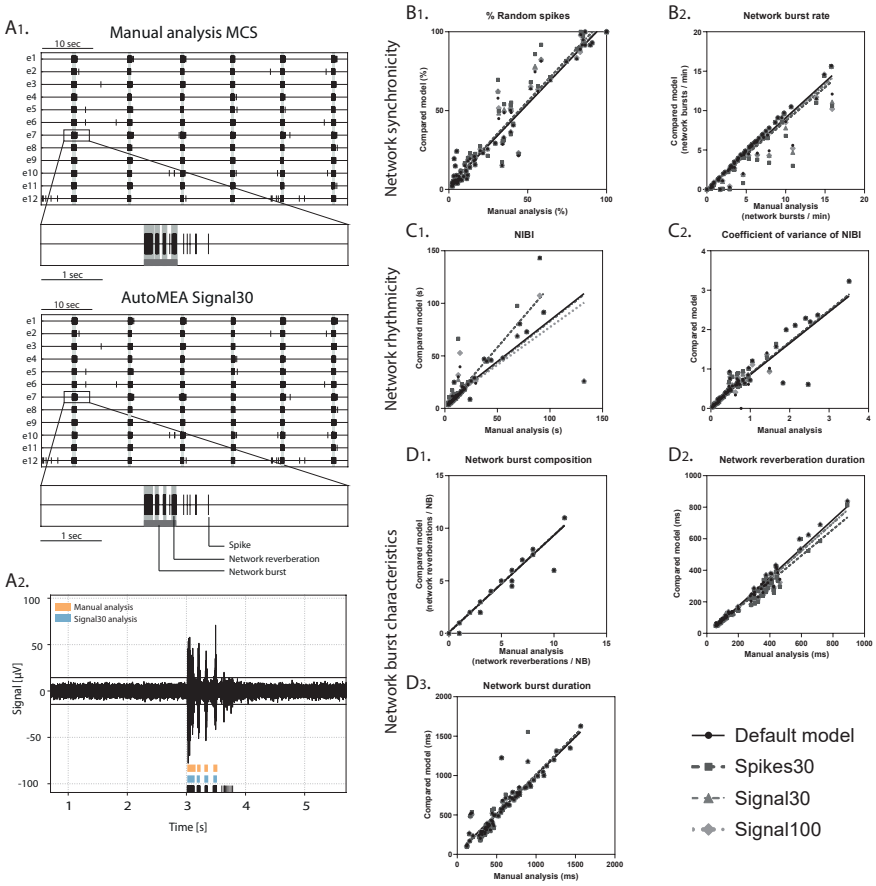


Figure 4.6: Validation of accurate parameter detection by the autoMEA models: A) 1. examples of raster plots and 5-second sections of a single electrode as detected by the Manual analysis in MCS (top) or signal30 autoMEA software (bottom). Black lines represent spikes, light gray bars overlaying raster plot represent reverberations, dark grey bar at the bottom of the zoom section represents the network burst. 2. raw data example of a single channel during a network burst with reverberation detection presented at the bottom of the graph by manual analysis MCS in orange (top) and signal30 autoMEA analysis in blue (middle) and spikes in black (bottom). B) correlation between outcome parameters for network synchronicity 1. % random spikes, 2. Network burst rate. C) correlation between outcome parameters for network rhythmicity 1. Network inter burst interval (NIBI), 2. Coefficient of variance of NIBI. D) correlation between outcome parameters for network burst characteristics 1. network burst composition, 2. network reverberation duration, 3. network burst duration. N = 81 wells.

of the patient-identified pathogenic RHEB-p.P37L variant [32], or a control virus.

First, we compared the network activity of control and RHEB-p.P37L neuronal networks, with bursts detected using the manual analysis to all autoMEA models at DIV14,

a time-point at which control hippocampal cultures show reverberating bursts synchronized across the network and in a rhythmic pattern. The averages of both genotypes were very similar, comparing the different autoMEA approaches to the manual analysis (Figure 4.7, for all outcome parameters, see Supplementary Figure 1). While there are slight differences in the exact numbers detected by the autoMEA software when compared to the manual analysis, these differences have the same directionality for both genotypes tested, e.g. the average network reverberation duration slightly reduces for both the control and RHEB-p.P37L group. Furthermore, performing statistics on the difference between the two groups revealed that all methods accurately detected previously identified phenotypes: a decrease in NBC (Figure 4.7 B) and an increase in network reverberation duration (Figure 4.7C). Parameters that did not manifest a phenotype through manual analysis, similarly remained non-significant when analyzed using the autoMEA models (Supplementary Figure 1).

To assess whether the software can also accurately detect bursts across the development of the culture, we included the analysis of the RHEB-p.P37L dataset at DIV7. At this time point, hippocampal cultures have not yet developed the reverberating network bursts and show more random spiking activity [14]. Similar to DIV14, we observed genotype averages very similar to the manual analysis for each parameter, and again, the directionality of change was the same for both genotypes (Figure 4.8, for all outcome parameters, see Supplementary Figure 2). Notably, also network bursts in younger cultures, without reverberations appear to be accurately detected by the autoMEA models, which we trained to detect reverberating bursts. Furthermore, statistical comparison of the groups showed that phenotypes were accurately detected in DIV7 cultures (Figure 4.8, for all outcome parameters, see Supplementary Figure 2).

CAMK2g-p.R292P

We included a second NDD model, that has not yet been extensively characterized using MEA. Patients with this mutation suffer from severe intellectual disability (ID), autism spectrum disorder (ASD), and general developmental delay [33, 34]. In this second validation experiment, we used a set of neuronal network activity recordings at DIV18, from primary hippocampal neuronal networks transduced at DIV1 with a virus expressing either CAMK2G wildtype (CAMK2G-WT), the previously published pathogenic variant of CAMK2G, CAMK2G-p.R292P [34] or a control virus. Manual analysis of this novel dataset presented multiple phenotypes. We observed a decrease in the firing rate in both CAMK2G-WT and CAMK2G-p.R292P cultures compared to cultures transduced with a control virus (Supplementary figure 3A). Furthermore, the expression of CAMK2G-p.R292P reduced network synchronicity as there was an increased percentage of random spikes (Supplementary figure 3B-C). Interestingly, we identified decreased rhythmicity of network bursts for CAMK2G-WT, but not CAMK2G-p.R292P cultures, as shown by the significant increase in the CoV-NIBI (Figure 4.9B). We further observed apparent phenotypes in the network burst characteristics, namely that the NBD significantly decreased in CAMK2G-p.R292P cultures, while the network reverberation duration increased (Figure 4.9C,E). CAMK2G-WT cultures displayed the opposite effect, an increased NBD while network reverberation duration significantly decreased. Finally, we observed an increase

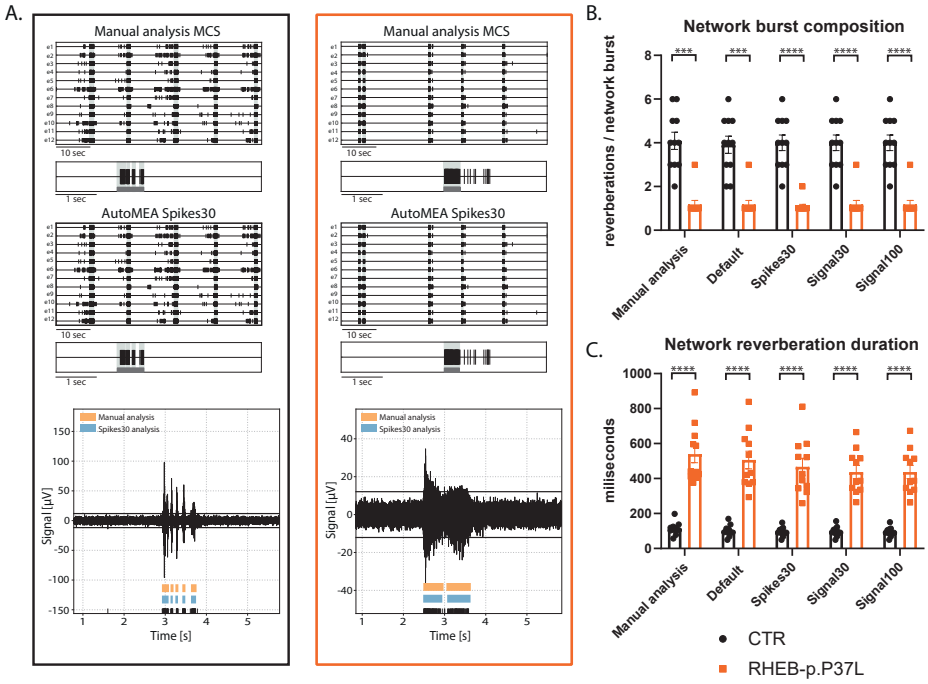


Figure 4.7: Validation of the detection of epilepsy-related phenotypes in a DIV14 set of the RHEB-p.P37L NDD model by the autoMEA software: A) Example raster plots with a 5-second section of a single electrode of a control (black, left) and RHEB-p.P37L (orange, right) well from manual MCS analysis and the spikes30 autoMEA model, and a raw data trace example of a single channel during a network burst for both genotypes at the bottom, black lines at the bottom represent spikes, orange bars (top) represent reverberations as detected with manual MCS analysis and the blue bars (middle) reverberations detected using the spikes30 autoMEA model. B) Comparison of the network burst composition for control and RHEB-p.P37L cultures detected using all different burst detection methods. C) Comparison of the network reverberation duration for control and RHEB-p.P37L cultures detected using all different burst detection methods. N = 11 wells/group. Student's t-test: ***p<0.0001, ****p<0.00001

in the NBC in the CAMK2G-WT cultures while this was drastically decreased in the CAMK2G-p.R292P cultures (Figure 4.9D). Also in this disease model, genotype averages were comparable between the manual analysis and the different autoMEA models and we could detect the same phenotypes in a novel dataset using the autoMEA models compared to the manual analysis.

In summary, the autoMEA model accurately detects phenotypes in hippocampal cultures of two different NDD models. Importantly, it detects phenotypes at multiple time points in the development of the cultures. This data shows that the autoMEA software is a reliable tool to analyze hippocampal MEA datasets. We did not observe striking differences between the performance of the different prediction models incorporated in the autoMEA software in the detection of NDD-related phenotypes.

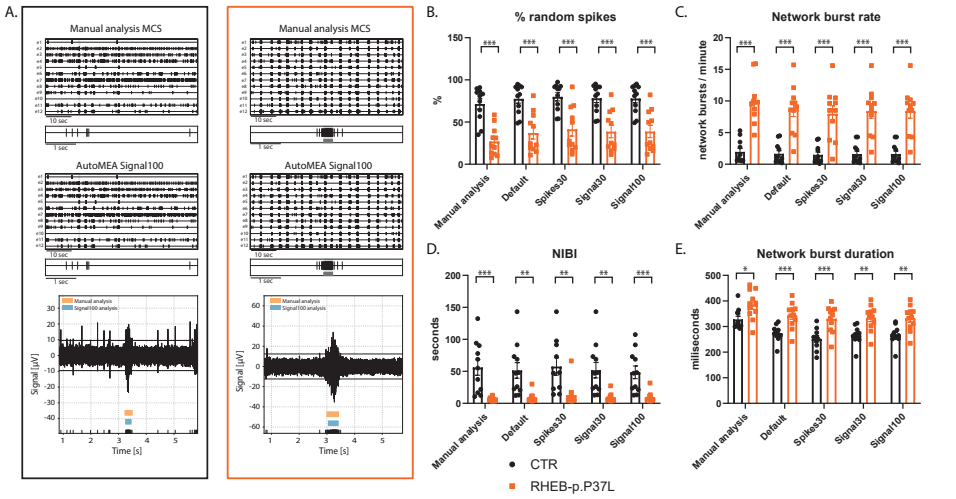


Figure 4.8: Validation of the detection of epilepsy-related phenotypes in a DIV7 set of the RHEB-p.P37L NDD model by the autoMEA software: A) Example raster plots with a 5-second section of a single electrode of control (black, left) and RHEB-p.P37L (orange, right) well from manual MCS analysis and the signal100 autoMEA model, and a raw data trace example of a single channel during a network burst for both genotypes at the bottom, black lines at the bottom represent spikes, orange bars (top) represent reverberations as detected with manual MCS analysis and the blue bars (middle) reverberations detected using the signal100 autoMEA model. B) Comparison of the %RS for control and RHEB-p.P37L cultures detected using all different burst detection methods. C) Comparison of the network burst rate for control and RHEB-p.P37L cultures detected using all different burst detection methods. D) Comparison of the NIBI for control and RHEB-p.P37L cultures detected using all different burst detection methods. E) Comparison of the network burst duration for control and RHEB-p.P37L cultures detected using all different burst detection methods. N = 11 wells/group. Student's t-test: *p<0.05, **p<0.01, ***p<0.001 ***p<0.0001, ****p<0.00001

Cortical data

To investigate if the performance of the model is specific to the hippocampal burst dynamics of the dataset that was used to generate the model, or if it can accurately detect bursts across datasets with different burst dynamics, we included a set of recordings from a cortical dataset at DIV14. While hippocampal cultures generate spontaneous reverberating network bursts, cortical cultures do not present this reverberating pattern (Figure 4.10). A cortical dataset of 18 wells was analyzed using the manual settings used to analyze the hippocampal data and analyzed using the autoMEA models. On top of that, the MaxInterval method parameters in the manual detection analysis were adjusted to more accurately detect bursts with cortical burst dynamics. To this extent, the maximum interspike intervals to start and end a burst were set to 100 ms, and the minimum interval between bursts was set to 200 ms, from here referred to as ISI100 analysis.
Similar to the hippocampal datasets, the detection of bursts with all its analyzed characteristics was very similar between the manual analysis and the autoMEA models for most of the wells. The adaptation of the MaxInterval parameters significantly affected the detection of the following parameters: as expected when the threshold for the ISI increases, network burst duration significantly increased, paired with a decrease in the %

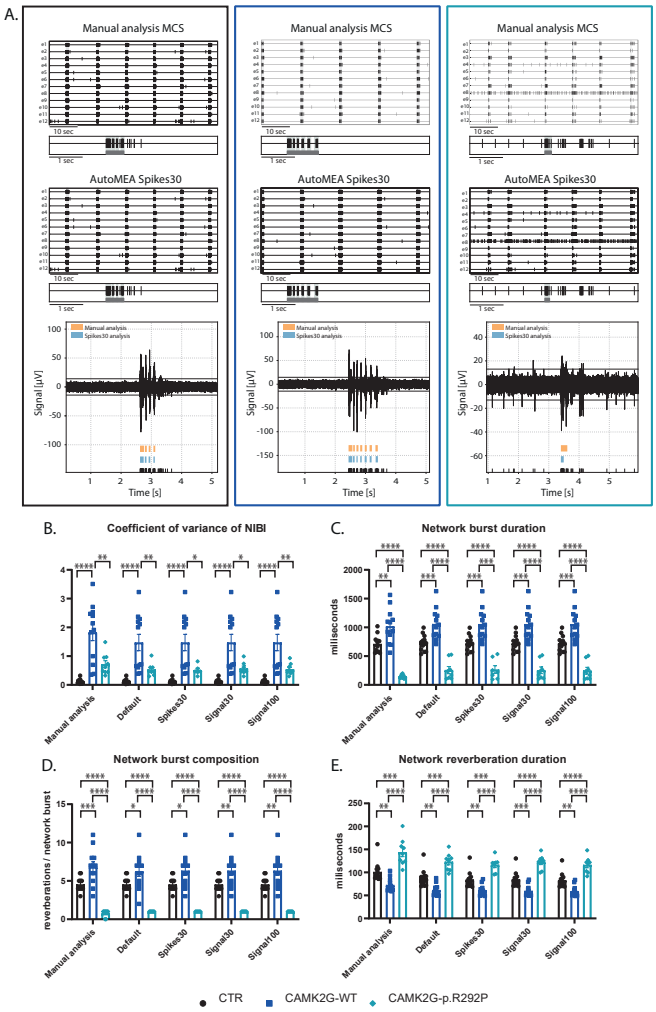


Figure 4.9: Validation of the detection of phenotypes in a DIV18 set of the CAMK2G-p.R292P NDD model by the autoMEA software: A) Example raster plots with a 5-second section of a single electrode of control (black, left) and CAMK2G-WT (dark blue, middle), and CAMK2G-p.R292P (light blue, right) well from manual MCS analysis and the spikes30 autoMEA model, and a raw data trace example of a single channel during a network burst for all genotypes at the bottom, black lines at the bottom represent spikes, orange bars (top) represent reverberations as detected with manual MCS analysis and the blue bars (middle) reverberations detected using the spikes30 autoMEA models. B) Comparison of the %RS for control, CAMK2G-WT, and CAMK2G-p.R292P cultures detected using all different burst detection methods. C) Comparison of the network burst rate for control, CAMK2G-WT, and CAMK2G-p.R292P cultures detected using all different burst detection methods. D) Comparison of the NIBI for control, CAMK2G-WT, and CAMK2G-p.R292P cultures detected using all different burst detection methods. E) Comparison of the network burst duration for control, CAMK2G-WT, and CAMK2G-p.R292P cultures detected using all different burst detection methods. N(control) = 13 wells, N(CAMK2G-WT) = 12, N(CAMK2G-p.R292P) = 12. One way ANOVA: *p<0.05, **p<0.01, ***p<0.001 ***p<0.0001, ****p<0.00001

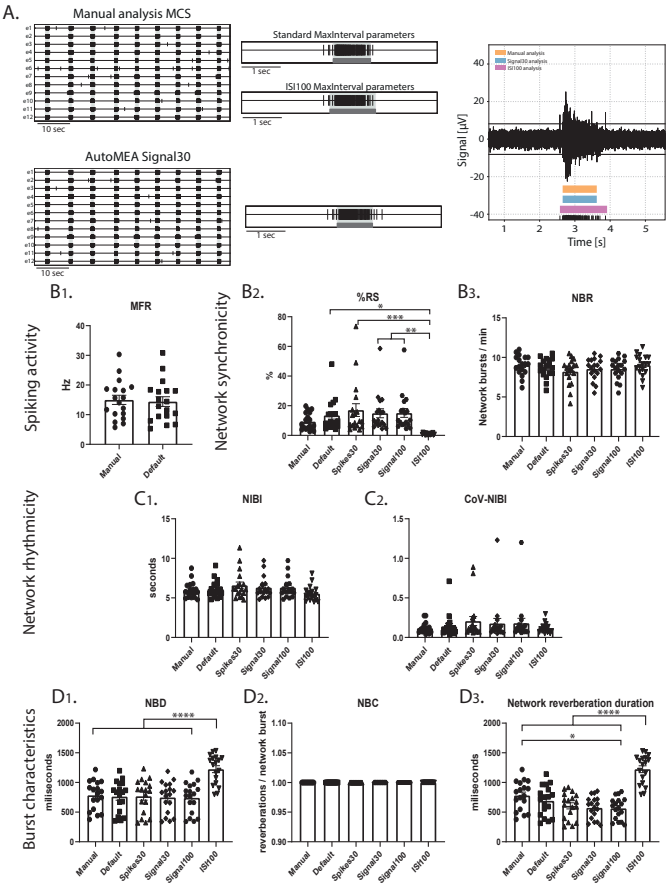


Figure 4.10: Testing the performance of autoMEA burst detection on a cortical dataset. A) Example raster plots with a 5-second section of a single electrode, analyzed using manual MCS analysis and the signal30 autoMEA model, and a raw data trace example of a single channel during a network burst for all genotypes at the bottom, black lines at the bottom represent spikes, orange bars (top) represent reverberations as detected with manual MCS analysis, the blue bars (middle) reverberations detected using the signal30 autoMEA models and the pink (bottom) the manual detection in MSC using ISI 100. B) Comparison of the MFR, %RS, and NBR using all different burst detection methods. C) Comparison of the NIBI and CoV-NIBI using all different burst detection methods. D) Comparison of the NBD and NBC and network reverberation duration using all different burst detection methods. N = 18 wells. One way ANOVA: *p<0.05, **p<0.01, ***p<0.001 ***p<0.0001, ****p<0.00001

RS, as more spikes were identified as part of the network burst (Figure 4.10). The other parameters regarding network burst frequency and rhythmicity were not significantly affected by increasing the ISI threshold, and as such, the autoMEA model accurately detected these outcome parameters in our cortical cultures.

Interestingly, the dataset also presented a clear outlier (Supplementary figure 4), which

appeared most obviously in the read-out parameters %RS and CoV-NIBI. Inspection of this well showed that some network bursts were not detected using the detection models, and most network bursts were missed when the spikes30 model was used. The amplitude of the spikes within these bursts appeared lower, however this was not quantified.

In summary, our findings demonstrate that our automated quantitative machine-learning based analysis software tool, autoMEA, is a reliable tool to analyze MEA datasets in a high-throughput manner, presumably without inter-experimenter bias. Moreover, the model's detection proficiency extends to effectively capture bursts with varying dynamics, showing its ability to generalize across different datasets.

4.3. DISCUSSION

In this project, we generated automated detection software that can be used for the analysis of neuronal network activity recorded using a multiwell MEA system. More specifically, our focus was on creating a package with the capability to accurately identify intricate burst dynamics inherent to hippocampal neuronal networks. With this approach, we additionally aimed to reduce the manual aspect of the analysis and improve the efficiency with which the data can be analyzed. We showed that the different detection models included in our autoMEA software can accurately detect network burst activity from the MEA signal, comparable to a manual data analysis, using a defined set of MaxInterval parameters. The outcome from the autoMEA models showed very strong correlations with the manual analysis. Additional scoring of the model's performances revealed that in most cases, the models could predict network bursts as well as the default MaxInterval parameters, in some cases performing even better than default as judged by an experienced experimenter. More importantly, the autoMEA models were able to identify the same phenotypes that were identified using manual analysis in two separate datasets for NDDs. Finally, the models could accurately detect bursts with different dynamics that appear throughout the maturation of a neuronal network, as was shown by the detection of bursts in a dataset of DIV7 neuronal networks.

MEA is a valuable tool for investigating neuronal network activity and is often used for disease modeling or toxicological assessments. However, burst detection remains a challenge in the field, as burst dynamics can vary depending on the type of cultures that are recorded [14, 35]. While previously, several MEA-analysis tools have been generated, they focused on the analysis and visualization of bursts with simpler network dynamics [15, 16, 17, 18]. In these models, bursts are detected based on the MaxInterval and/or logISI burst detection methods that have not been optimized for the detection of reverberations. Here, we presented a model that accurately detected reverberations based on the MaxInterval method but using adaptive parameters to optimize reverberation detection, a parameter that is sensitive to spiking noise.

With autoMEA software, we also provide experimenters with an open-source user-friendly software package. While the software that is provided with the commercial hardware outputs timestamp files that need to be post-processed to extract relevant parame-

ters, our model does not require any need for coding expertise or manual data processing to post-process the output into quantifiable outcome parameters that are relevant to describe neuronal network dynamics. Besides outputting timestamp files, it automatically generates an additional output file with the network parameters describing network synchronicity, rhythmicity, and burst characteristics. Furthermore, one can input multiple recording files into the software, letting it analyze multiple recordings at the same time. Additionally, manual analysis requires the experimenter to actively adjust settings during the analysis of each file, while with autoMEA software, the analysis is performed completely autonomously without the experimenter's input. Running the software may take from minutes up to a few hours, depending on the size of the dataset. Therefore autoMEA software enhances the throughput of MEA-data analysis. For a demonstration of the user-friendliness of autoMEA, see Supplemental Information.

Our MEA package is an open source package that can be completely adapted depending on the user's needs. The key feature is that the machine learning models can be fine-tuned or retrained using new datasets. This may be preferred when datasets with different burst dynamics than the murine hippocampal cultures are analyzed. Researchers can build upon the current dataset, which could enhance the accuracy of the model to detect bursts with different burst dynamics, and importantly, can further reduce any experimenter bias as the model is now trained based on the analysis of an experienced researcher. By blinding the experimenter to bursts presented during the training, we tried to ensure the objectivity of the burst quality metric introduced in this study.

In this study, the machine learning model was generated to specifically detect the more complex burst dynamics observed in hippocampal neuronal networks, in which bursts generally consist of multiple reverberations. To broaden the usefulness of the autoMEA software, we showed that our package could also accurately detect many of the outcome parameters in a cortical dataset. However, as burst dynamics are different in cortical cultures, the MaxInterval parameter threshold for the ISI with which this type of data is analyzed is typically higher [7], which affected significantly the outcome parameters %RS and the duration of network bursts and reverberations. For different types of data, it may be necessary to retrain the models with a specific dataset, however, the autoMEA software could still be used without retraining if a predetermined set of MaxInterval parameters is known. In this case, the software can be run using the default method, similar as to what was done using the manual MCS analysis settings in the default method of the autoMEA software in this paper. We did not observe obvious differences between the performance of the different prediction models (spikes30, signal30, signal100). Both using binary spike input and real-valued signal, the machine learning approach was trained to accurately detect parameters describing network dynamics. The only small difference was identified in the detection of the NIBI, wherein the NIBI was increased using the spikes30 model compared to default, while it was reduced in the signal models. However, these deviations were small and did not affect the detection of phenotypes in our NDD models. Therefore, we consider all autoMEA models suitable for the analysis of MEA-data.

The autoMEA machine learning approach for detecting bursts from signal or spikes demonstrated robust generalization across diverse datasets. We introduced a customized accuracy metric that evaluates the difference between bursts detected by manually set MaxInterval Parameters and those predicted by our machine learning models, enabling precise performance assessment. Despite being trained on a modest dataset derived from a limited set of recordings, these models accurately replicate the experimenter's MaxInterval Parameter selections for analysis. It's noteworthy that all the machine learning models examined in this study are based on simple convolutional neural networks that are well established in the machine learning community and straightforward to implement. Despite their simplicity, all the machine learning models assessed in this study have shown impressive accuracy. Notably, the autoMEA package facilitates easy fine-tuning of the used machine learning models with additional data or their substitution with more advanced architectures, ensuring adaptability to evolving research needs.

With recent technological advancements that allow the development of neuronal networks derived from iPSCs, the MEA system has become more popular as a functional readout in disease modeling studies using stem cell methods [19, 9]. Interestingly, in multiple disorders, the appearance of reverberations, otherwise referred to as fragmented bursts or super bursts, was identified as a phenotype [20, 36, 21, 23]. Therefore, we believe that our software can be of interest to a broader audience. The flexibility of our software allows users to use the models trained with the datasets presented in this paper, but also retrain it using their own dataset to optimize detection in datasets with different burst dynamics. Additionally, adding training data onto the current dataset may increase the ability of the software to analyze more complex or diverse datasets, and may result in better convergence of the model onto a dataset with varying burst dynamics. Thus, we provided here an effective software tool for multi-well MEA analysis that is user-friendly, high-throughput, and adaptable to the researcher's preferences.

4.4. METHODS

4.4.1. MEA DATA COLLECTION

MEA recordings of primary hippocampal neurons

Primary hippocampal and cortical neuronal cultures were prepared from embryonic day (E) 16.5 FvB/NHsD wild-type mice according to the procedure previously described [31] [37]. Neurons were plated in a multiwell multi-electrode array (MEA) plate with an epoxy base (Multichannel Systems, MCS GmbH, Reutlingen, Germany) in a density of 35,000 neurons / well. Cultures were maintained in neurobasal medium (NB, GIBCO) supplemented with 2% B27, 1% penicillin/streptomycin and 1% glutamax (NB+++) and placed in an incubator at 37 ◦ C with 5% CO2.

Each MEA well is embedded with 12 PEDOT-coated gold electrodes of 100 μm in diameter and 1 reference electrode. Recording electrodes are arranged in a 4x4 grid, spaced 700 μm apart.

Twice weekly, neuronal network activity of the cultures was recorded after which one-third of the medium in each well was replaced with fresh NB+++.

MEA plates were recorded using the Multiwell-MEA headstage in a recording chamber at 37 C with 5% CO2. Recordings were started after 10 minutes of acclimatization in the MEA set-up. Channels with excessive noise (above +/- 15mV) were excluded from the recording. Neuronal activity was recorded for 10 minutes at a sampling rate of 10 kHz and the signal was filtered with a 4th order low-pass filter at 3.5kHz and 2nd order high-pass filter at 100 Hz [32].

Manual MEA data analysis using the MCS software
MEA data was manually analyzed using the MultiChannel System software package. Analysis was performed on the full 10-minute recording period. Baseline noise was calculated as the average of 2x200 ms segments without activity at the start of the analysis period, and a threshold of +/- 5 SD from baseline was used to detect spikes. Reverberations were detected using the MaxInterval method that is incorporated into the MCS software. For the dataset used in this study, the most ideal parameters for reverberation detection were identified as:

Max. interval to start = 15 ms
Max. interval to end = 20 ms
Min. interval between = 25 ms
Min. duration = 20 ms
Min. number of spikes = 5

For the cortical dataset, manual analysis of the same wells was run with the settings adapted to Max. interval to start and end a burst = 100 ms and Min. interval between bursts at 200 ms.
Whenever at least two-thirds of the active channels in a well participated in synchronized activity, of which at least half were simultaneously active, it was classified as a network reverberation.
Output from the MCS Software was then further processed using a custom-written script in MATLAB R2021a. Reverberations were combined into bursts when the interval to the next reverberation was <300ms, and similarly, network reverberations were combined into network bursts when the interval to the next network reverberation was <300 ms.

Using the custom-written processing script in MATLAB, multiple outcome parameters were extracted from the data that can describe the network development and dynamics in the culture. We calculated 8 different outcome parameters that could be classified into 4 categories: 1) Spiking activity described by the mean firing rate (MFR), 2) Network synchronicity, described by network burst rate (NBR), and the % random spikes, i.e. spikes that are not part of a network burst (%RS). 3) Network rhythmicity, described by the network interburst interval (NIBI) and more specifically the coefficient of variance thereof (CoV-NIBI). 4) Network burst characteristics, to which the network burst duration (NBD), network reverberation duration, and network burst composition (NBC: network reverberations/network burst) are descriptive.

4.4.2. MACHINE-LEARNING AUTOMATION

The ultimate goal of MEA data analysis is to quickly and robustly detect bursts for large amounts of measured data. While the ideal MaxInterval parameters are hard to unify across the dataset, due to the spiking noise, it is still possible to adapt them manually to the different levels of noise. These manual adjustments can be automatized if one is able to develop an algorithmic mapping from MEA signal (or spikes) to the parameters during the processing of the dataset. Machine learning is a powerful tool that is able to approximate complex multivariable functions as well as to generalize well under the influence of noise. In our approach, we chose to use MEA data, such as processed signals and spikes, as input to a supervised neural network, trained to output optimal MaxInterval parameters for each data sample. This way, we allow for the parameters to be continuously adapted without the constant attention of a human operator. In this approach, the model developed closely mimics what an experimenter does when analyzing MEA data: it finds the parameters that help extract the best burst configuration from the dataset. More specifically, the parameters predicted by the model are used as input for the MaxInterval method to detect reverberations, which are then used as input to precited bursts, network reverberations, and network bursts, using fixed (user-defined) parameters.

Below we describe how we generated a dataset to be used to train and test the models developed, and details about the methods implemented.

Dataset generation

In order to train and test the models in this work, we generated a dataset in which we selected five-second windows of MEA activity, during which bursts occur, and together with the experimenter expert, the values of the first three MaxInterval parameters (Max. interval to start, Max. interval to end, and Min. interval between bursts) are adjusted until an optimal burst detection set is obtained. This process is repeated multiple times, adding at each iteration one sample to the dataset - for this work, a total of 797 samples were generated, using recordings for different systems at different DIVs. For each selected window, all the data useful to train and evaluate the developed models is saved. For the sampling rate considered, 10kHz, a five-second window corresponds to 50 thousand timestamps, which is why most of the data is saved as arrays with lengths equal to 50 thousand. Specifically, MEA signal is saved a float array, while spikes and bursts as binary arrays, with an element equal to zero in case there is no spike/bursts activity occurring at the correspondent timestamp, and equal to one in case there is activity. Finally, the MaxInterval parameters are saved as an integer array with length equal to three - since we are just interested in the first three parameters. Afterwards, part of the original dataset was post-processed, to get variables in the form of input/output used by the different models developed, and divided into Training/Validation/Test sets. In details, the signal arrays were normalized between 0 and 1, and all temporal arrays (signal, spikes and bursts) were averaged by either 30 or 100 timestamps. In the end, we have three different inputs for each approach considered. In Table 4.2 we show the various inputs and outputs used for each model developed, assigning a name for each specific model.

Table 4.2: Relation of input, output and name of every model developed in this work.

Approach	Input (length)	Output (length)	Name
Parameter Prediction	Binary spike array (1667)	Float parameter array (3)	spikes30P
	Float signal array (1667)		signal30P
	Float signal array (500)		signal100P

Prediction of MaxInterval parameters

The models implemented consist of convolutional neural networks that receive as input MEA data, and map it into three of the five MaxInterval parameters. Different models were developed based on the different input choices, as described in the Dataset Generation section. All models were developed using the Tensorflow/Keras framework. The models were trained in a supervised learning regime, where a loss function - Mean Square Error in this case - is defined to calculate an error between the convolutional networks' predicted output and the target output - the manually selected MaxInterval parameters. This error is then used to update the internal parameters of the model until the predicted output converges to the target. This iterative procedure used to optimize the model parameters is called training. During training, the model calculates different loss values for the samples taken from the Training Set and the Validation Set, with the main difference being that the model parameters are just updated based on the loss retrieved from the Training Set only. Moreover, the iterative training process is divided into epochs, where one epoch is an instance for which the model used the totality of the Training/Validation Set.

While designing a convolutional neural network many hyperparameters have to be defined, such as the architecture of the network, which optimizer is used to change the internal parameters, and how many samples are used before updating the parameters. In order to find the best hyperameter for each model implemented - depending on the input type, we used a package called hyperas, which works within the keras framework and makes it possible to define a set of values for each hyperparameter, and scan which combination of these values return the best models, based on the final loss value. Then, using hyperas, we trained each of the three models one hundred times using different hyperameters values, and post-selected three best cases for each one, based on the behavior of the training and validation loss.

The three best cases for each model were trained again, now calculating a new metric to characterize the accuracy of the model. Accuracy is a metric used to quantify the performance of a machine learning model, however, it is just meaningful when it is used for classification, when the output is a discrete value correspondent to a class, and the objective is to distinguish between different classes. In the Parameter Prediction approach, the models developed are performing what is called regression, when the output is a real value, used to estimate the value of a variable. However, the error that defines a distance between the predicted MaxInterval parameters and the target values is not enough by itself to define how the models are performing. The final goal of the automation developed in this work is to obtain optimal bursts, independently by the combination of

MaxInterval parameters used to detect these bursts. We defined a custom accuracy metric, which compares the predicted bursts - in this case, the bursts detected using the predicted MaxInterval parameters, and the target bursts - the bursts detected using the target MaxInterval parameters. Considering the binary representation of the burst arrays, the custom accuracy is defined as

$$A = \frac{\sum_i |b_p[i] - b_t[i]|}{\sum_i b_t[i]}, \tag{4.1}$$

where $b_p[i]$ is the binary burst array, obtained using the predicted parameters, at index i, and $b_t[i]$ is the binary burst array obtained using the target parameters, at index i. This metric quantifies how many timestamps the bursting state differs between the bursts detected using predicted and target parameters, normalized by the number of timestamps in which the target bursts are active (equal to one). The normalization is necessary, given the sparse nature of the binary burst arrays, to avoid high accuracy values in cases where the predicted bursts are a full-zero array (no burst activity).

Calculation of the custom accuracy while training a model takes a considerable amount of time, since for each sample used a burst has to be detected and compared to the target one. That is why we just perform the custom accuracy calculation for the three best-performing cases for each model. Then, from the new training procedure, one best model for each input type is chosen, based on both the loss and the custom accuracy, and is trained five more times to obtain averaged values of loss/accuracy and test its consistency.

To further quantify the model performance, we defined a new metric, called Burst Quality, using the test set (never seen by the model), in which we detect bursts using both the MaxInterval parameters predicted by the ML-model, and those used as default by the experimenter expert. Both sets of bursts are shown to the expert together with the correspondent signal and spikes, and the expert votes on which burst detection better represents that in the specific time trace. For this we built a GUI that shows difference signals/spikes figures, randomly shuffling the position/color with which bursts detected using predicted/default parameters are plotted.

4.4.3. MODEL VALIDATION

To assess whether the model could accurately detect phenotypes in models for NDDs that were identified using the manual analysis in the MCS software, datasets of two different NDD models were used: RHEB-p.P37L and CAMK2G-p.R292P. The RHEB-p.P37L pathogenic variant has previously been identified in focal cortical dysplasia type 2 and is associated with severe epilepsy [30, 31], and has been extensively characterized using the Multi-electrode array [32]. The CAMK2G-p.R292P pathogenic variant has been identified in patients with severe intellectual disability [33, 34]. These disorders were modeled through a lentivirally induced expression of the RHEB-p.P37L, CAMK2G-WT or CAMK2-p.R292P genes, compared to transduction with a control virus. Recordings from different days *in vitro* were included in this study to verify the accurate detection of bursts throughout the development of the culture. Manual and autoMEA analysis were done using data from DIV7 and DIV14 for the RHEB-p.P37L model, and at DIV18 for

the CAMK2G-p.R292P model. Additionally, the convergence of the model onto a cortical dataset was tested using a wild-type dataset of cortical data recorded at DIV14.

4.4.4. STATISTICS

Statistical analyses were performed using GraphPad Prism 5 (GraphPad Software, Inc., CA, USA). Burst detection accuracy was tested using the Chi-square test. The normality of the data was assessed using the Shapiro-Wilk test. The correlation for each outcome parameter comparing the model analysis to the manual analysis was analyzed using Pearson's r or the non-parametric alternative if the normality assumption was not met, and the linear relationship was plotted using simple linear regression. Statistical analysis of disease phenotypes was performed using a Student's t-test (RHEB-p.P37L data), or One-way ANOVA (CAMK2g-p.R292P and cortical data). For all statistical analyses, alpha was set at 0.05. The specific tests used for each experiment are specified in the figure legends or the results section. Values are represented as averages ± SEM. Sample sizes for each experiment are indicated in the figure legends.

4.5. SUPPLEMENTAL TEXT

4.5.1. MODELS' ARCHITECTURE AND HYPERPARAMETERS

From the hyperameter optimization step while training the machine learning models used in this work, it was possible to choose three best models for each input type, for which the training was repeated calculating the custom accuracy. Then, one single model is chosen for each input type, based on the highest value of custom accuracy achieved. An overview of the final models' hyperameters is shown in Table 4.3.

Table 4.3: (DRAFT - have to recheck every hyperameter in the office pc) Hyperparameters for the best model for each input type. CL refers to the Convolutional Layer and FC to the Fully Connected Layer.

		Spike30	Signal30	Signal100
CL1	Filters	128	128	128
	Kernel size	3	7	7
CL2	Filters	64	64	64
	Kernel size	5	7	7
	Max Pooling	Yes	No	No
	Dropout	0.33	0.18	0.18
CL3	Filters	-	16	16
	Kernel size	-	9	9
FC1	Units	64	32	32
	Dropout	0.12	0.62	0.62
Optimizer		RMSProp	SGD	SGD
Learning Rate		1E-6	1E-4	1E-4
Batch size		2	4	4
Trainable parameters		6865731	921299	323795

4.5.2. AUTOMEA FULL ANALYSIS METHOD

In this section, we showcase an example of how to use the autoMEA package in practice. The easiest way to use autoMEA is to perform a full analysis of a series of datasets and corresponding wells, using one of the machine learning models available with the package for burst detection.

The standard workflow consists of preparing a *csv* file containing datasets and corresponding wells to be analyzed. The structure to be followed is shown below in the example file called `datasets_and_wells.csv`.

Input file example

```
#dataset, wells
dataset_1.h5, A1 A2 B2
dataset_2.h5, B1 C2
```


This file is used to analyze the wells `A1`, `A2` and `B2` from `dataset_1.h5`, and wells `B1` and `C2` from `dataset_2.h5`.

If the *csv* file, the datasets to be analyzed, and the machine learning model saved as a *h5* file are present in the same folder, a script to run the analysis can be:

Python Code

```
import automea

# create automea analysis object
am = automea.Analysis()

# define machine learning model to use, and load it
am.model_name = 'signal30.h5'
am.loadmodel()

# define which output will be saved
am.analysis_params['save_stats'] = True
am.analysis_params['save_net_bursts'] = True

# run analysis
am.analyze_dataset('datasets_and_wells.csv')
```

The analysis is performed using the model called `signal30.h5`, and a statistics file and a network bursts file are produced as output, as specific by setting the correspondent items of the `analysis_params` attribute as `True`.

The network bursts are saved as *csv* file with information about the datasets and well analyzed, and information about each network burst. An example of the network bursts file structure is shown below, with ellipses used to omit less significant columns present in the file.

Network bursts output file example

```
Dataset, Well Label, Start time[µs], Duration[µs],Spike Count,...
dataset_1.h5, A1, 981800, 395800, 498, ...
dataset_2.h5, C2, 465000, 120500, 112, ...
\end{verbatim}
```

The statistics output file contains several higher-order statistical quantities that are calculated during the analysis. An example of the file structure is shown below, using again ellipses to omit some of the columns present in the file.

Statistics output file example

```
Dataset, Well Label, Fir. Rate[Hz], Stray spikes[%], ...
dataset_1.h5, A1, 7.8, 11.92, ...
dataset_2.h5, B1, 8.2, 12.33, ...
```

This example shows how easy and practical it is to use autoMEA as a ready-to-use analysis package. One of the main goals of the package is to place itself as an accessible alternative to researchers with limited coding skills. At the same time, more advanced functionalities can be accessed, and modified depending on users' needs. A series of tutorials showcasing how to use autoMEA to perform tailored analysis of MEA datasets can be found on https://automea.readthedocs.io.

4.5.3. SUPPLEMENTARY FIGURES

Supplementary figure 1

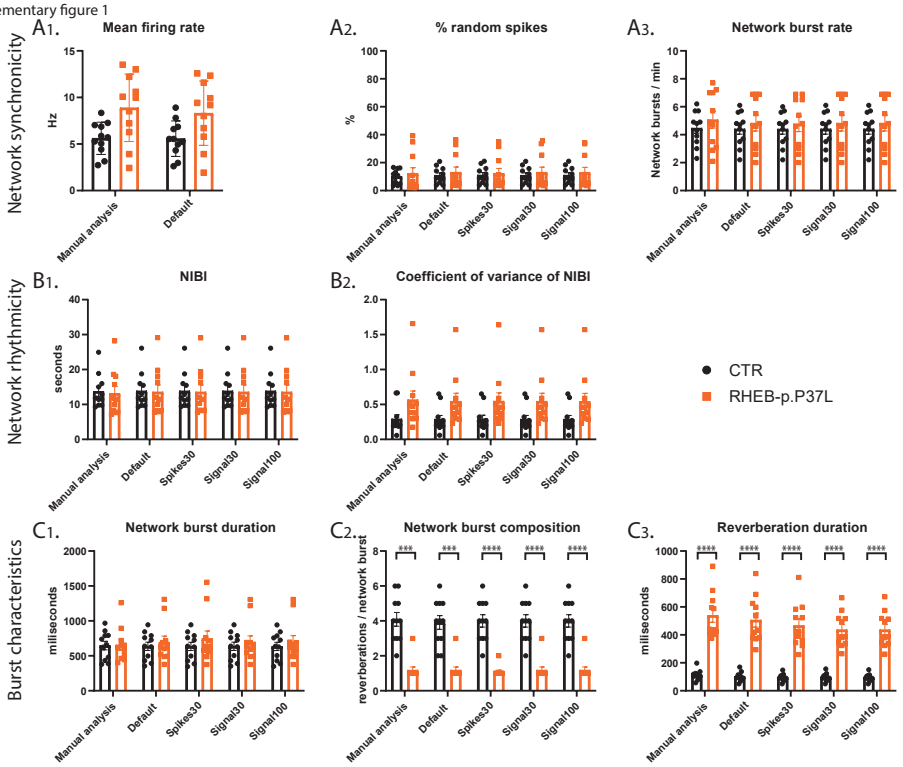


Figure 4.11: Validation of the detection of epilepsy-related phenotypes in a DIV14 set of the RHEB-p.P37L NDD model for all outcome parameters by the autoMEA software: A) detection by manual analysis and autoMEA for outcome parameters describing spiking activity and network synchronicity. B) detection by manual analysis and autoMEA for outcome parameters describing network rhythmicity. C) detection by manual analysis and autoMEA for outcome parameters describing burst characteristics. N = 11 wells/group. Student's t-test: *p<0.05, **p<0.01, ***p<0.001 ***p<0.0001, ****p<0.00001

Supplementary figure 2

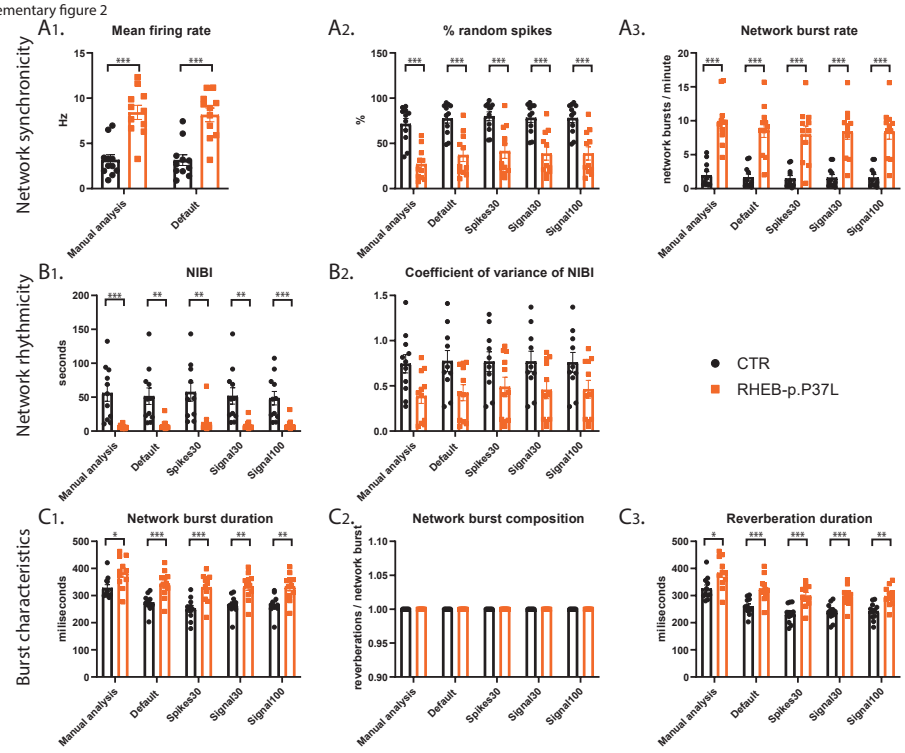


Figure 4.12: Validation of the detection of epilepsy-related phenotypes in a DIV7 set of the RHEB-p.P37L NDD model for all outcome parameters by the autoMEA software: A) detection by manual analysis and autoMEA for outcome parameters describing spiking activity and network synchronicity. B) detection by manual analysis and autoMEA for outcome parameters describing network rhythmicity. C) detection by manual analysis and autoMEA for outcome parameters describing burst characteristics. N = 11 wells/group. Student's t-test: *p<0.05, **p<0.01, ***p<0.001 ***p<0.0001, ****p<0.00001

Supplementary figure 3

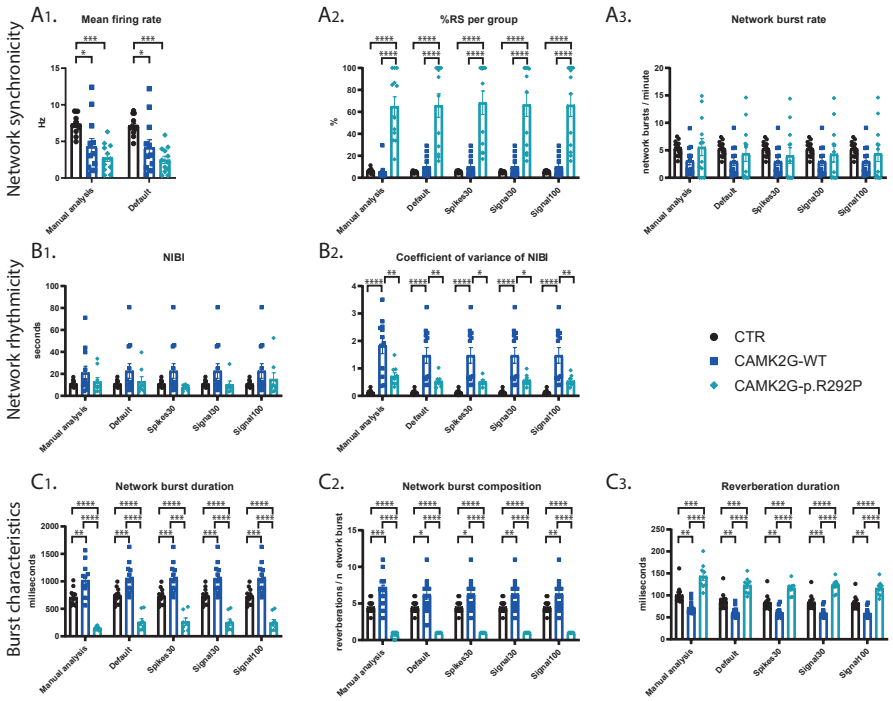


Figure 4.13: Validation of the detection of epilepsy-related phenotypes in a DIV18 set of the CAMK2G-p.R292P NDD model for all outcome parameters by the autoMEA software: A) detection by manual analysis and autoMEA for outcome parameters describing spiking activity and network synchronicity. B) detection by manual analysis and autoMEA for outcome parameters describing network rhythmicity. C) detection by manual analysis and autoMEA for outcome parameters describing burst characteristics. N(control) = 13 wells, N(CAMK2G-WT) = 12, N(CAMK2G-p.R292P) = 12. One way ANOVA: *p<0.05, **p<0.01, ***p<0.001 ***p<0.0001, ****p<0.00001

4

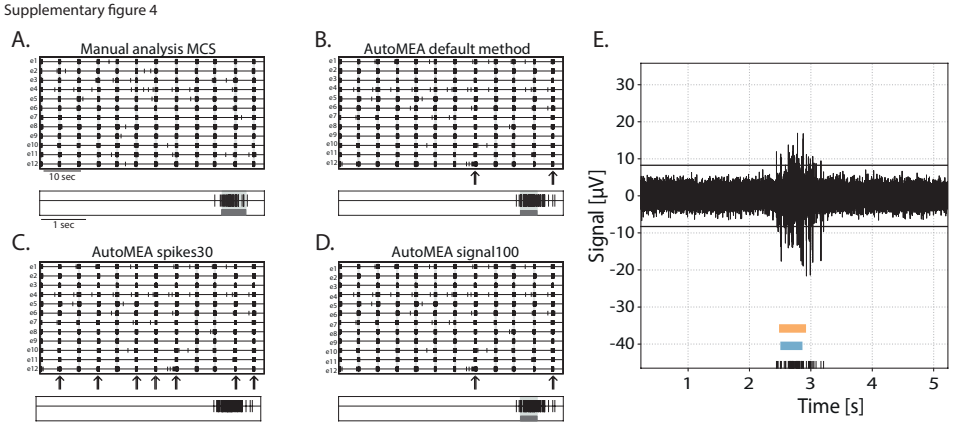


Figure 4.14: Example of outlier neuronal network in the cortical dataset A-D) Rasterplot and 5 second zoom in of a single electrode of spikes and bursts detected by manual analysis (A), autoMEA default method (B), spikes30 model (C), and signal100 model (D). Black arrows represent network bursts that are not detected in using the autoMEA software. E) raw trace of a burst that was not accurately detected using the autoMEA software. black lines at the bottom represent detected spikes, blue bar (bottom) represents burst as detected using the manual analysis, orange bar (top) represents burst as detected using the default method.

REFERENCES

[1] Vinicius Hernandes et al. “autoMEA: Machine learning-based burst detection for multi-electrode array datasets”. In: *Frontiers in Neuroscience* 18 (2024), p. 1446578.

[2] Jeffry S Isaacson and Massimo Scanziani. “How inhibition shapes cortical activity”. In: *Neuron* 72.2 (2011), pp. 231–243.

[3] V Pasquale et al. “Self-organization and neuronal avalanches in networks of dissociated cortical neurons”. In: *Neuroscience* 153.4 (2008), pp. 1354–1369.

[4] Jaap Van Pelt et al. “Long-term characterization of firing dynamics of spontaneous bursts in cultured neural networks”. In: *IEEE Transactions on Biomedical Engineering* 51.11 (2004), pp. 2051–2062.

[5] CA Thomas Jr et al. “A miniature microelectrode array to monitor the bioelectric activity of cultured cells”. In: *Experimental cell research* 74.1 (1972), pp. 61–66.

[6] Ian L Jones et al. “The potential of microelectrode arrays and microelectronics for biomedical research and diagnostics”. In: *Analytical and bioanalytical chemistry* 399 (2011), pp. 2313–2329.

[7] Michela Chiappalone et al. “Burst detection algorithms for the analysis of spatio-temporal patterns in cortical networks of neurons”. In: *Neurocomputing* 65 (2005), pp. 653–662.

[8] S Martinoia et al. “In vitro cortical neuronal networks as a new high-sensitive system for biosensing applications”. In: *Biosensors and Bioelectronics* 20.10 (2005), pp. 2071–2078.

[9] Marta Cerina, Maria Carla Piastra, and Monica Frega. “The potential of in vitro neuronal networks cultured on Micro Electrode Arrays for biomedical research.” In: *Progress in Biomedical Engineering* (2023).

[10] Michela Chiappalone et al. “Dissociated cortical networks show spontaneously correlated activity patterns during in vitro development”. In: *Brain research* 1093.1 (2006), pp. 41–53.

[11] Britt Mossink et al. “Human neuronal networks on micro-electrode arrays are a highly robust tool to study disease-specific genotype-phenotype correlations in vitro”. In: *Stem cell reports* 16.9 (2021), pp. 2182–2196.

[12] CR Legendy and M Salcman. “Bursts and recurrences of bursts in the spike trains of spontaneously active striate cortex neurons”. In: *Journal of neurophysiology* 53.4 (1985), pp. 926–939.

[13] Valentina Pasquale, Sergio Martinoia, and Michela Chiappalone. “A self-adapting approach for the detection of bursts and network bursts in neuronal cultures”. In: *Journal of computational neuroscience* 29 (2010), pp. 213–229.

[14] Paul Charlesworth et al. “Quantitative differences in developmental profiles of spontaneous activity in cortical and hippocampal cultures”. In: *Neural development* 10 (2015), pp. 1–10.

[15] Raha M Dastgheyb, Seung-Wan Yoo, and Norman J Haughey. “MEAnalyzer–a Spike Train Analysis Tool for Multi Electrode Arrays”. In: *Neuroinformatics* 18.1 (2020), pp. 163–179.

[16] Michel Hu et al. “Mea-toolbox: An open source toolbox for standardized analysis of multi-electrode array data”. In: *Neuroinformatics* 20.4 (2022), pp. 1077–1092.

[17] Luca Leonardo Bologna et al. “Investigating neuronal activity by SPYCODE multi-channel data analyzer”. In: *Neural Networks* 23.6 (2010), pp. 685–697.

[18] Vito Paolo Pastore et al. “S pi C o D yn: A Toolbox for the Analysis of Neuronal Network Dynamics and Connectivity from Multi-Site Spike Signal Recordings”. In: *Neuroinformatics* 16 (2018), pp. 15–30.

[19] Fraser P. McCready et al. “Multielectrode Arrays for Functional Phenotyping of Neurons from Induced Pluripotent Stem Cell Models of Neurodevelopmental Disorders”. In: *Biology* 11.2 (2022). ISSN: 2079-7737. DOI: 10.3390/biology11020316. URL: https://www.mdpi.com/2079-7737/11/2/316.

[20] Kartik S Pradeepan et al. “Calcium-Dependent Hyperexcitability in Human Stem Cell–Derived Rett Syndrome Neuronal Networks”. In: *Biological Psychiatry Global Open Science* 4.2 (2024), p. 100290.

[21] Michele Gabriele et al. “KMT2D haploinsufficiency in Kabuki syndrome disrupts neuronal function through transcriptional and chromatin rewiring independent of H3K4-monomethylation”. In: *bioRxiv* (2021). DOI: 10.1101/2021.04.22.440945. eprint: https://www.biorxiv.org/content/early/2021/04/23/2021.04.22.440945.full.pdf. URL: https://www.biorxiv.org/content/early/2021/04/23/2021.04.22.440945.

[22] Eline JH Van Hugte et al. “SCN1A-deficient excitatory neuronal networks display mutation-specific phenotypes”. In: *Brain* 146.12 (2023), pp. 5153–5167.

[23] Monica Frega et al. “Neuronal network dysfunction in a model for Kleefstra syndrome mediated by enhanced NMDAR signaling”. In: *Nature communications* 10.1 (2019), p. 4928.

[24] Renato Durrer et al. “Automated tuning of double quantum dots into specific charge states using neural networks”. In: *Physical Review Applied* 13.5 (2020), p. 054019.

[25] Rouven Koch et al. “Adversarial Hamiltonian learning of quantum dots in a minimal Kitaev chain”. In: *Physical Review Applied* 20.4 (2023), p. 044081.

[26] Gunasekaran Manogaran and Daphne Lopez. “A survey of big data architectures and machine learning algorithms in healthcare”. In: *International Journal of Biomedical Engineering and Technology* 25.2-4 (2017), pp. 182–211.

[27] Anton Kocheturov, Panos M Pardalos, and Athanasia Karakitsiou. “Massive datasets and machine learning for computational biomedicine: trends and challenges”. In: *Annals of Operations Research* 276 (2019), pp. 5–34.

[28] Vinicius Hernandes, Anouk M. Heuvelmans, Valentina Gualtieri, Dimphna H. Meijer, Geeske M. van Woerden, and Eliska Greplova. *autoMEA GitLab*. https://gitlab.com/QMAI/papers/autoMEA. 2024. URL: %7Bhttps://gitlab.com/QMAI/papers/autoMEA%7D.

[29] Vinicius Hernandes, Anouk M. Heuvelmans, Valentina Gualtieri, Dimphna H. Meijer, Geeske M. van Woerden, and Eliska Greplova. *autoMEA documentation*. https://automea.readthedocs.io. 2024. URL: %7Bhttps://automea.readthedocs.io%7D.

[30] Margot RF Reijnders et al. "Variation in a range of mTOR-related genes associates with intracranial volume and intellectual disability". In: *Nature communications* 8.1 (2017), p. 1052.

[31] Martina Proietti Onori et al. "RHEB/mTOR hyperactivity causes cortical malformations and epileptic seizures through increased axonal connectivity". In: *PLoS Biology* 19.5 (2021), e3001279.

[32] Anouk M. Heuvelmans et al. "Modeling mTORopathy-related epilepsy in cultured murine hippocampal neurons using the multi-electrode array". In: *Experimental Neurology* 379 (2024), p. 114874. ISSN: 0014-4886. DOI: https://doi.org/10.1016/j.expneurol.2024.114874. URL: https://www.sciencedirect.com/science/article/pii/S0014488624002000.

[33] Joep De Ligt et al. "Diagnostic exome sequencing in persons with severe intellectual disability". In: *New England Journal of Medicine* 367.20 (2012), pp. 1921–1929.

[34] Martina Proietti Onori et al. "The intellectual disability-associated CAMK2G p. Arg292Pro mutation acts as a pathogenic gain-of-function". In: *Human mutation* 39.12 (2018), pp. 2008–2024.

[35] Daniel A Wagenaar, Jerome Pine, and Steve M Potter. "An extremely rich repertoire of bursting patterns during the development of cortical cultures". In: *BMC neuroscience* 7 (2006), pp. 1–18.

[36] Nina Doorn et al. "Breaking the Burst: Unveiling Mechanisms Behind Fragmented Network Bursts in Patient-derived Neurons". In: *bioRxiv* (2024), pp. 2024–02.

[37] G Banker and K Goslin. "Developments in neuronal cell culture." In: *Nature* 336.6195 (1988), pp. 185–186.

5

QDsim: A user-friendly toolbox for simulating large-scale quantum dot devices

We introduce QDsim, a python package tailored for the rapid generation of charge stability diagrams in large-scale quantum dot devices, extending beyond traditional double or triple dots. QDsim is founded on the constant interaction model from which we rephrase the task of finding the lowest energy charge configuration as a convex optimization problem. Therefore, we can leverage the existing package CVXPY, in combination with an appropriate powerful solver, for the convex optimization which streamlines the creation of stability diagrams and polytopes. Through multiple examples, we demonstrate how QDsim enables the generation of large-scale dataset that can serve a basis for the training of machine-learning models for automated tuning algorithms. While the package currently does not support quantum effects beyond the constant interaction model, QDsim is a tool that directly addresses the critical need for cost-effective and expeditious data acquisition for better tuning algorithms in order to accelerate the development of semiconductor quantum devices.

The results of this chapter have been published as: V. Gualtieri, C. Renshaw-Whitman, **V. Hernandes**, E. Greplova, "QDsim: A user-friendly toolbox for simulating large-scale quantum dot devices", SciPost Physics Codebases **46** (2025) [1]

5.1. INTRODUCTION

Quantum dots (QDs) have emerged as a particularly promising quantum computing platform [2, 3, 4, 5, 6, 7]. These semiconducting systems, that operate by trapping charge carriers in potential wells called "dots", have been the subject of extensive research due to their scalability potential and the relative simplicity to fabricate them, that leverages already existing techniques in the semiconductor industry.

The scalability of quantum dot-based qubits is considered a cornerstone for practical quantum computation. At the same time, the complexity of these systems scales with the number of quantum dots, and it poses a significant challenge in establishing and maintaining the desired electron occupancy across the array.

Furthermore, with increasing device size, the task of manually tuning each quantum dot becomes impractical. The scaling of these systems necessitates an automated approach to tuning. In this context, artificial intelligence (AI) emerges as a highly promising solution. AI algorithms have the potential to learn and adapt to the complex variety of quantum dot behaviors, automating the tuning process with efficiency and precision.

However, the efficacy of AI depends on the availability of extensive datasets that capture the diverse operational regimes of quantum dot arrays. These datasets are critical for training robust machine learning. The scarcity of such data is a significant bottleneck in the advancement of AI applications within this field [8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

In response to these challenges, we present `QDsim`, a novel computational framework designed to simulate the electrostatic environment of quantum dot arrays efficiently. Our approach reduces the complexity of the simulation problem to a convex optimization task, offering an efficient and user-friendly solution. `QDsim` is implemented as a open-source Python package, providing a flexible tool for quantum dot array design and large-scale data generation for machine learning (ML) applications [18].

The most important feature of `QDsim` is its ability to generate charge stability diagrams, which are essential for understanding the operational regimes of quantum dot arrays. These diagrams show the connection between gate voltages and charge configurations, and are characterized by a tessellation of the voltage space into polytopes. The geometry of these polytopes provides insights into the charge configuration of the quantum dot system. While other simulators of quantum dot arrays exist, `QDsim` package offers unprecedented flexibility in geometry of the device in term of placement of dots, gates and sensors. Additionally, we demonstrate a significant speed in the charge stability diagram generation that allows for rapid simulations of 100+ quantum dots.

By enabling the rapid generation of charge stability diagrams for large-scale quantum dot devices, `QDsim` serves as a foundational tool for creating the vast datasets required for machine learning training. Its ability to simulate complex quantum dot arrays and produce charge stability diagrams is a step towards the future where AI-driven automatization becomes the standard for quantum device tuning.

The paper is structured as follows. In Section 5.2 we introduce the theory behind the model: constant capacitance model and Coulomb polytopes. In Section 5.3 we discuss key `QDsim` classes and provide detailed explanation of the package functionalities. In Section 5.4 we discuss relevant examples for the `QDsim`. We provide detailed discussion of default device designs and their possible customization as well as instructions on how to a design and simulate completely custom architectures.

5.2. FORMULATION OF THE ELECTROSTATIC MODEL: CHARGE STABILITY DIAGRAMS

In this section, we describe the theoretical foundation of the `QDsim` package: the constant-interaction (or constant-capacitance) model applied to quantum dot arrays.

Charge stability diagrams are visual manifestations of the Coulomb blockade effect, a quantum phenomenon that occurs in small conducting or semiconducting structures, such as quantum dots. At the quantum scale, electrons are not free to flow into and out of a quantum dot without restriction; instead, they are influenced by the Coulomb force from other electrons within the dot. When an electron is added to a quantum dot, it increases the energy of the system due to this repulsive force. If the energy required to add another electron exceeds the thermal energy of the system, the dot will not take on any additional electrons until the external conditions (such as gate voltage) are altered. This leads to a blockade of charge transfer, which is observable as discrete jumps in the conductance through the quantum dot.

Experimentally, charge stability diagrams are obtained by varying the voltages on the electrostatic gates that control the quantum dots and measuring the resulting conductance. For multiple dots devices, these diagrams exhibit a characteristic pattern, each corresponding to a stable number of electrons within the quantum dots. The boundaries between these regions represent points of charge degeneracy, where the number of electrons on a dot can change.

In our model, we represent these phenomena within the classical framework of the constant interaction [19]. In this model, each quantum dot is considered a conductor with capacitive coupling to other dots and to electrostatic gates. The charge stability diagrams emerge from this model as a tessellation of the gate voltage space, where each region corresponds to a stable charge configuration. These configurations are the ground states of the system's free energy function.

By simulating charge-stability diagrams, `QDsim` provides a powerful tool for efficiently simulating the behavior of quantum dot arrays.

5.2.1. DERIVATION OF THE CONSTANT-CAPACITANCE MODEL ENERGY EQUATION

In the constant interaction model, each quantum dot is considered a conductor with capacitive coupling to other dots and to electrostatic gates [20] [21].

The electrostatic characteristics of the system with N_D dots and N_G gates are captured by two mutual capacitances matrices:

- the dot-to-dot mutual capacitance matrix, in which each element $\widetilde{C_{ij}^{DD}}$ represents the capacitive coupling between dot i and dot j, and the diagonal elements represent the dot's self capacitance; and
- the dot-to-gate mutual capacitance matrix, in which each element $\widetilde{C_{ij}^{DG}}$ represents the capacitive coupling between dot i and gate j.

The dot-to-dot mutual capacitance matrix $\widetilde{C^{DD}}$ requires $N_D(N_D-1)/2$ values due to the symmetrical relation of the mutual capacitances, i.e. the capacitance between

element i and element j is the same between element j and element i. No symmetry relations are present in the dot-to-gate mutual capacitance matrix $\widetilde{C^{DG}}$, therefore it is defined by $N_D N_G$ values.

Capacitance, C, relates the charge state Q, i.e. the charge contained on the conductor, to the electrostatic potential, V via $Q = CV$. For multiple conductors, Q and V are column vectors $\mathbf{Q}$ and $\mathbf{V}$, and C is a matrix $\mathbf{C}$. However, in order for the relation

$$\mathbf{Q} = \mathbf{CV} \tag{5.1}$$

to hold true in matrix form, some distinctions must be made. Here we first introduce the total system's mutual capacitance $\widetilde{C}$, which is obtained by stacking the dot-to-dot and dot-to-gate capacitance matrices in the following way:

$$\widetilde{\mathbf{C}} = \begin{bmatrix} \widetilde{\mathbf{C}}_{DD} & \widetilde{\mathbf{C}}_{DG} \\ \widetilde{\mathbf{C}}_{DG}^T & 1 \end{bmatrix}. \tag{5.2}$$

Matrix $\widetilde{\mathbf{C}}$ satisfies the condition

$$Q_i = \sum_j \widetilde{C_{ij}}(V_i - V_j), \tag{5.3}$$

which is different from the aforementioned $\mathbf{Q} = \mathbf{CV}$. Here the indexes i, j run from 1 to $N_D + N_G$. V_i represents the voltage applied to the i-conductor. V_j represents the voltage applied to the j-conductor. These conductors are both dots and gates, accounting for a total of $N_D + N_G$ conductors. Specifically, in our notation, a general index i running from 1 to N_D will account for the dots, while the same index running from $N_D + 1$ to $N_D + N_G$ will account for the gates.

Upon manipulation, the relation $\mathbf{Q} = \mathbf{CV}$ holds when $\mathbf{C}$ is the Maxwell matrix [22], defined as:

$$C_{ij} = \delta_{ij} \sum_k \widetilde{C}_{ik} + (1 - \delta_{ij})(-\widetilde{C}_{ij}), \tag{5.4}$$

where δ_{ij} is the Kronecker delta.

Given that all self-capacitances $\widetilde{C_{ii}}$ are positive, the Maxwell matrix is strictly diagonally dominant [23], ensuring its invertibility via the Levy-Desplanques theorem [24].

Distinguishing between dot and gate properties, we express $\mathbf{Q}$, $\mathbf{V}$ and $\mathbf{C}$ as:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_D \\ \mathbf{Q}_G \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} \mathbf{V}_D \\ \mathbf{V}_G \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \mathbf{C}_{DD} & \mathbf{C}_{DG} \\ \mathbf{C}_{DG}^T & \mathbf{C}_{GG} \end{bmatrix}. \tag{5.5}$$

The capacitance relation in Equation (5.1) then becomes:

$$\begin{bmatrix} \mathbf{Q}_D \\ \mathbf{Q}_G \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{DD} & \mathbf{C}_{DG} \\ \mathbf{C}_{DG}^T & \mathbf{C}_{GG} \end{bmatrix} \begin{bmatrix} \mathbf{V}_D \\ \mathbf{V}_G \end{bmatrix}. \tag{5.6}$$

The free energy function F of the system is given by:

$$F = U - W = \frac{1}{2} [\mathbf{Q}_D^T, \mathbf{Q}_G^T] \begin{bmatrix} \mathbf{V}_D \\ \mathbf{V}_G \end{bmatrix} - \mathbf{V}_G^T \mathbf{Q}_G. \tag{5.7}$$

Assuming $\mathbf{Q}_D = e\mathbf{N}_D$, where e is the elementary charge with sign (+1 for holes, −1 for electrons), we can rewrite the free energy function in terms of $\mathbf{N}_D$ (the charge configuration that we want to obtain) and $\mathbf{V}_G$ (the gate voltages that we want to tune). Omitting gate self-energies, we rewrite the free energy from Equation (5.7) as

$$\begin{aligned} F(\mathbf{N}_D;\mathbf{V}_G) = &\frac{1}{2}\big(e^2\mathbf{N}_D^T\mathbf{C}_{DD}^{-1}\mathbf{N}_D \\ &-2e\mathbf{N}_D^T\mathbf{C}_{DD}^{-1}\mathbf{C}_{DG}\mathbf{V}_G \\ &+\mathbf{V}_G^T\mathbf{C}_{DG}^T\mathbf{C}_{DD}^{-1}\mathbf{C}_{DG}\mathbf{V}_G\big). \end{aligned} \tag{5.8}$$

For the remainder of the paper, we will adopt units such that $|e| = 1$, and define the charging-energy matrix $\mathbf{E}_C = \mathbf{C}_{DD}^{-1}$. It is worth noting that we will need to take into account the sign of the charge, which will be negative for electrons, and positive for holes simulations.

5.2.2. GROUND STATES AND COULOMB POLYTOPES IN V-SPACE

In this section we show that the ground states of the free energy function define the polytopes in the voltage space. We define the ground-state energy $F_{GS}(V_G)$ and the corresponding occupation numbers $GS(V_G)$ as

$$F_{GS}(\mathbf{V}_G) = \min_{\mathbf{N}\in\mathbb{Z}^{N_D}} F(\mathbf{N};\mathbf{V}_G), \tag{5.9}$$

$$GS(\mathbf{V}_G) = \{\mathbf{N}\in\mathbb{Z}^{N_D} \mid F(\mathbf{N},\mathbf{V}_G) = F_{GS}(\mathbf{V}_G)\}. \tag{5.10}$$

The set of voltages for which a given occupation $\mathbf{N}_0$ is a ground state, $GS^{-1}(\mathbf{N}_0)$, is determined by the condition that $F(\mathbf{N}_0,\mathbf{V}_G) \preceq F(\mathbf{N}_0 + t,\mathbf{V}_G)$ for all $t \in \mathbb{Z}^{N_D}$. This leads to the the following description of a convex polytope

$$(t^T\mathbf{E}_C\mathbf{C}_{DG})\mathbf{V}_G \preceq \frac{1}{2}t^T\mathbf{E}_C t + t^T\mathbf{E}_C\mathbf{N}_0, \tag{5.11}$$

where $\preceq$ denotes element-wise inequality.

Convex polytopes can be defined as an intersection of a finite number of half-spaces. This definition is called a half-space representation or H-description [25]. Inequality (5.11) represents the H-description of the coulomb polytopes.

A direct consequence of the inequality (5.11) is that the regions in V_G-space admitting a particular occupation $\mathbf{N}$ as a ground state form convex polytopes. We can also prove that two polytopes sharing an interior point $\mathbf{V}_0$ must coincide, implying that two states $\mathbf{N}_1$ and $\mathbf{N}_2$ are degenerate if and only if $\mathbf{N}_1 - \mathbf{N}_2 \in \text{Null}(\mathbf{C}_{DG}^T\mathbf{E}_C)$.

To prove this statement, we can consider two polytopes which share a point $\mathbf{V}_0$ interior to each, therefore all the inequalities in (5.11) hold strictly. Suppose that two ground-states, $\mathbf{N}_1$ and $\mathbf{N}_2$, are admitted for the point $\mathbf{V}_0$, which is in the interior of the respective polytopes. Then there exists an open ball B of finite radius $\eta \in \mathbb{R}^{++}$ such that all the points in $B_\eta(\mathbf{V}_0)$ are also in both polytopes. Within this ball, by assumption the occupations $\mathbf{N}_1$ and $\mathbf{N}_2$ remain ground states, therefore have the same energy. Then for all $\delta\mathbf{V} \in B_\eta(0)$, the following holds:

$$F(\mathbf{N}_1, \mathbf{V}_0 + \delta\mathbf{V}) = F(\mathbf{N}_2, \mathbf{V}_0 + \delta\mathbf{V}) \tag{5.12}$$

$$\begin{aligned}\frac{1}{2}(\mathbf{N}_1^T\mathbf{E}_C\mathbf{N}_1 - \mathbf{N}_2^T\mathbf{E}_C\mathbf{N}_2) - (\mathbf{C}_{DG}\mathbf{V}_0)^T\mathbf{E}_C(\mathbf{N}_1 - \mathbf{N}_2)\\ = (\mathbf{C}_{DG}\delta\mathbf{V})^T\mathbf{E}_C(\mathbf{N}_1 - \mathbf{N}_2)\end{aligned} \tag{5.13}$$

The left hand side is equal to 0 when $\delta\mathbf{V} = 0$. Hence, the right-hand side vanishes independently of $\delta\mathbf{V}$. Thus $N_1^T\mathbf{E}_C\mathbf{C}_{DG} = N_2^T\mathbf{E}_C\mathbf{C}_{DG}$. It follows that $F(\mathbf{N}_1, \mathbf{V}_0) = F(\mathbf{N}_2, \mathbf{V}_0)$ for all $\mathbf{V}_0 \in \mathbb{R}^{N_G}$. Further, any two states are degenerate if and only if $\mathbf{N}_1 - \mathbf{N}_2 \in \text{Null}(\mathbf{C}_{DG}^T\mathbf{E}_C)$ [26].

In this section, we have demonstrated that the task of identifying Coulomb diamonds can be reformulated through convex optimization. The established convexity enables the framing of Equation (5.8)'s minimization as a convex problem. This approach leads to the determination of ground states as the sought-after solution.

5

5.3. QDSIM PACKAGE

`QDsim` is a Python package that bridges the gap between theoretical frameworks and practical quantum dot device simulations. This versatile tool comprises three essential classes: `QDDevice` (quantum dot device), `QDSimulator` (quantum dot simulator), and `CapacitanceQuantumDotArray` (capacitance quantum dot array). All the code is available in a public repository on GitLab [27].

5.3.1. THE QUANTUM DOT DEVICE CLASS: QDDEVICE

The `QDDevice` class is a key element for device definition. It focuses on translating design of the device into capacitance matrices, specifically addressing dot-to-dot and dot-to-gate mutual capacitance matrices. This class is responsible for defining key parameters related to the device's geometry and design. Users can specify the number of dots, the number of gates, the locations of the dots, as well as the dot-to-dot $\widetilde{\mathbf{C}_{DD}}$ and dot-to-gate $\widetilde{\mathbf{C}_{DG}}$ mutual capacitance matrices.

Upon initialization, the `QDDevice` class provides an empty object, which users can then populate with the desired device characteristics. A range of standard design options is available, each tailored to specific device configurations. These standard options include the following methods:

- `one_dimensional_dots_array`: This option configures a line of dots with individual gate control, where users can define the number of dots, their locations, the dots' self-capacitance, the average dot-to-gate capacitance, if the dots are equal (i.e. they all have the same self-capacitance), if the gates are equal (i.e. if the dot-to-gate capacitance is the same for every couple in which the gate directly controls the dot), and the strength of the crosstalk interaction.
- `bi_dimensional_10_dots_array` : It sets up a 2D array of 10 dots with individual gate control, allowing customization of the device properties and capacitances as listed in the previous case.

- `crossbar_array_shared_control`: This option creates a crossbar array of dots with shared control. Users can specify the number of dots per side of the square lattice, and other device characteristics as in the previous case.

The `QDDevice` class features several attributes, including device type (for plotting purposes), the number of dots, the number of gates, dot self-capacitance, dot locations, dot-to-dot mutual capacitance matrices, and dot-to-gate mutual capacitance matrices.

Additionally, `QDDevice` offers methods for setting dot locations, the number of gates, custom dot-to-dot mutual capacitance matrices, and custom dot-to-gate mutual capacitance matrices. It also provides functionality for automatically evaluating dot-to-dot and dot-to-gate mutual capacitance matrices based on dot locations, assuming individual gate control. These methods facilitate the definition and customization of quantum dot devices and enable users to configure the device to their specific requirements, making it a versatile tool for simulations. Here, we provide an overview of the key methods offered:

- `set_physical_dot_locations`: This method allows users to assign dot locations to the `QDDevice` object. By specifying the coordinates (x, y) of each dot in the device, users can precisely define the spatial arrangement of quantum dots.

- `set_dot_dot_mutual_capacitance_matrix`: With this method, users can assign a custom dot-to-dot mutual capacitance matrix to the `QDDevice` object. This level of customization enables precise modeling of the capacitance interactions between quantum dots.

- `set_dot_gate_mutual_capacitance_matrix`: Similarly, users can define a custom dot-to-gate mutual capacitance matrix using this method. The dot-to-gate capacitance matrix plays a crucial role in simulating the interactions between quantum dots and gate electrodes.

- `evaluate_dot_dot_mutual_capacitance_matrix`: This method calculates the dot-to-dot mutual capacitance matrix of the `QDDevice` based on the dot locations. It employs a distance-based model to compute the capacitance interactions, taking into account the arrangement of quantum dots.

- `evaluate_dot_gate_mutual_capacitance_matrix`: This method is used to compute the dot-to-gate mutual capacitance matrix based on the dot locations. It assumes that each gate corresponds to a dot in only controls that dot, making it suitable for certain device configurations.

The package's versatility extends to device visualization, with a plotting method `plot_device` that allows users to create graphical representations of the device, including optional sensors and dot labels. The generated plots are exportable in various image formats, such as PDF and PNG, allowing flexibility in saving visual representations. Device attributes can also be exported to JSON files with the `save_to_json` method, providing a structured format for documentation. These JSON files can be easily imported and utilized using the `load_from_json` class method.

INDEXING

We begin by explaining the indexing conventions used for `one_dimensional_dots_array`, `crossbar_array_shared_control`, and potential custom configurations.

In the context of `QDsim` package, each dot and gate is associated with a distinct index, designated as i and j respectively. These indices range from 0 to $N_D - 1$ for dots and $N_G - 1$ for gates. For instance, the matrix element $\tilde{\mathbf{C}}_{ij}^{DD}$ denotes the mutual capacitance between dot i and dot j. This notation is consistently applied across the various mutual capacitance matrices within the package.

In the standard design templates provided, the indexing scheme of the dots and gates is readily represented through a plot of the device, via the `plot_device` method. Within the `one_dimensional_dots_array` configuration, the dots and their corresponding gates are sequentially numbered from left to right, from 0 to $N_D - 1$. Given the one-to-one correspondence between dots and gates in this arrangement, gate indices are not explicitly depicted; however, they adhere to the same left-to-right numbering convention, extending from 0 to $N_G - 1$, where N_G equals N_D in this case.

The `crossbar_array_shared_control` design has a distinct indexing pattern, as can be seen in Figure 5.1.

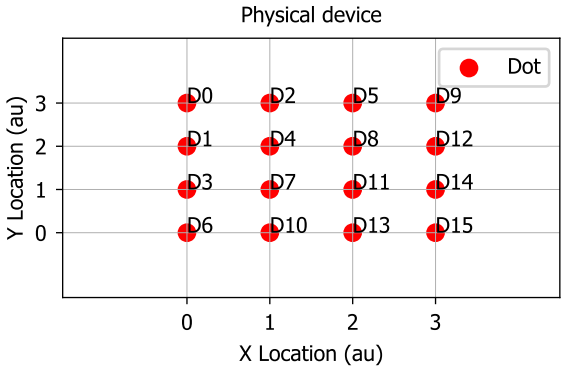


Figure 5.1: Example of the indexing of the dots in the crossbar array design. The 'D' stands for 'dot', the adjacent number represents the index.

Here, dots are numbered starting from the upper left corner, progressing downwards, and subsequently ascending diagonally towards the right. This process is repeated, descending from the leftmost point for one step and ascending in the rightward diagonal direction.

This convention ensures that increasing dot indices are governed by the same gate. The gates themselves are arranged diagonally and are numbered from the upper left to the lower right corner.

It is of course possible for users to modify our template configurations, but when doing so it is essential to adhere to the above described indexing convention such that correct capacitance matrices are generated.

5.3.2. THE SIMULATOR CLASS: QDSimulator

The `QDSimulator` class is a fundamental component of the `QDsim` package, and serves as the user's interface for simulating quantum dot charge stability diagrams.

The `QDSimulator` class is built atop the `CapacitanceQuantumDotArray` class. The `CapacitanceQuantumDotArray` class is utilized to construct and solve the convex optimization problem that characterize the quantum dot array. It will be covered in details in the next section. At this point, the important distinction to be made is that the `QDSimulator` class is a wrapper of the `CapacitanceQuantumDotArray` class. Therefore the user will interact directly only with the `QDSimulator` class, leaving the `CapacitanceQuantumDotArray` class working in the background.

When creating an instance of `QDSimulator`, the user has the discretion to specify the type of simulation—either 'Electrons' or 'Holes'. By default, the system assumes an 'Electrons' simulation.

The key attributes of the `QDSimulator` class are the following:

- `_qd_device`: This attribute represents the quantum dot device to be simulated and must be an instance of the `QDDevice` class.
- `_physics_to_be_simulated`: Determination whether the device consists of 'Electrons' or 'Holes'.
- `_variable_gate_index_1` and `_variable_gate_index_2`: These represent the indices of the gates to be scanned, i.e. which gates will be shown in the charge stability diagram axes. For simulations of the charge stability diagram, only a pair of gates can be simultaneously scanned.
- `_voltage_ranges`: This attribute sets the minimum and maximum voltages for the x- and y-axes, associated with gates indexed by `_variable_gate_index_1` and `_variable_gate_index_2`, respectively.
- `_sensor_locations`: This attribute determines the spatial coordinates (x, y) of each sensor within the device layout, which must be set via the class method `set_sensor_locations`. While the simulation itself, framed as a minimization problem, is independent of sensors, they are incorporated for more realistic visualization purposes. Specifically sensors allow visualization of realistic potential and current values and they would be monitored in an experiment.

The core method of this class is`simulate_charge_stability_diagram`. Central to the `QDsim` package, this function bridges the representation of the quantum dot device, defined by the `QDDevice` object, to the framework of the constant interaction model. The method utilizes a `CapacitanceQuantumDotArray` object, which serves as the powerhouse driving the entire simulation.

Users can choose their preferred convex optimization solver, including options like the open-source SCIP or the licensed MOSEK, for use within CVXPY. This higher-level interface seamlessly integrates with either MOSEK or SCIP as its back-end solvers, offering flexibility in solver selection.

Following the philosophy of flexibility, this method allows the specification of the gates to be scanned, with the number of probe points on both x- and y-axes, and an individual voltage range for each. Furthermore, the fixed voltage approach ensures a fixed potential for non-scanned gates, while the gates' voltages can be individually defined for more granular control. All the details will be described in Section 5.4 by taking advantage of some use cases.

A beneficial feature is the saving mechanism: The resulting arrays of occupation, potential, and current can be stored using the associated file path parameters. This ensures that simulation data can be revisited or shared without the need to rerun computations.

The final feature of the `QDSimulator` class is its inbuilt capability to visualize the simulated results. Through its integrated plotting methods, users can render charge stability diagrams.

The `plot_charge_stability_diagrams` method of the class is designed to create a visual representation of the charge stability diagram. Its features include:

- **Colormap Customization**: By default, the method employs the 'RdPu' colormap. However, users can modify the colormap using the `cmapvalue` argument.
- **Noise Inclusion**: The method allows for the introduction of Gaussian, white, or pink noise to the plots, enhancing the realism of simulations. They can be added by using the boolean `gaussian_noise`, `white_noise`, and `pink_noise` arguments.
- **Potential vs. Current Mapping**: While the default visualization mode displays the current map, there exists an option to showcase the potential map instead by setting the boolean argument `plot_potential` to True.
- **Custom Noise Parameters**: Users have the liberty to specify parameters for Gaussian, white, and pink noise using the `gaussian_noise_custom_params`, `white_noise_custom_params`, and `pink_noise_custom_params` arguments. In the absence of user-defined values, default settings are applied.
- **Saving Plots**: If desired, the generated visualizations can be saved to a predetermined file path, set via the `save_plot_to_filepath` argument.

5.3.3. THE POWERHOUSE OF THE PACKAGE: THE CAPACITANCEQUANTUMDOTARRAY CLASS

The `CapacitanceQuantumDotArray` class acts as the core computational engine of our framework. It completes the task of defining the problem parameters, establishing the correct environment, and running the actual simulation. This class operates predominantly in the background; users typically interact with higher-level interfaces such as `QDDevice` and `QDSimulator` and do not directly engage with `CapacitanceQuantumDotArray`.

The `CapacitanceQuantumDotArray` class is rooted in convex optimization techniques, particularly leveraging the CVXPY package. It aims to minimize the system's free energy, defined in Equation 5.7.

To obtain the system's ground state defined in Equation 5.9, we manipulated the free energy expression (Equation 5.8) in terms of $\mathbf{N}_D$ and $\mathbf{V}_G$, where $\mathbf{N}_D$ delineates the dot

occupations. The free energy undergoes minimization concerning $\mathbf{N}_D$, ensuring that $\mathbf{N}_D$ remains an integer vector.

For simplicity in calculations and units, the class assumes the unit charge $|e|$ as 1, meaning the free energy is denoted in eV.

Here we present a brief overwiew of the key methods:

- `select_solver`: This method allows to choose a solver for the convex optimization problem. Available options include 'MOSEK' and 'SCIP'. This method takes in input the solver selected by the user while interacting with the `QDSimulator` class. The user never access the methods of the `CapacitanceQuantumDotArray` class directly.

- `probe_voltage_space`: It explores the entire voltage space and determines the ground state for each point, returning both the dot occupations and associated energy.

- `_find_ground_state`: For a specific voltage point, this method identifies the system's ground state.

- `_set_up_convex_optimization_problem`: This method prepares the convex optimization problem, serving as a foundation for `_find_ground_state`.

- `_evaluate_maxwell_matrices`: Here, the system's Maxwell capacitance matrix is determined, acting as a precursor for the optimization setup.

5.4. EXAMPLES

In this section we highlight several common use-cases of QDsim, some of which corresponding to recently experimentally achieved devices [7] [6].

5.4.1. THE DOUBLE DOT DEVICE

Let us begin with a fundamental example of quantum dot device: double quantum dot. We begin with a `QDDevice` object in order to specify the physical parameters of the device. The `one_dimensional_dots_array` method significantly streamlines this initialization. By setting the `n_dots` parameter to 2, the system is automatically configured as a double dot device. The physical parameters of this default configuration can be seen by calling the `print_device_info` method, as shown below.

Python Code

```
from qdsim import QDDevice, QDSimulator

# create a quantum dot device object
qddevice = QDDevice()
# double dot
qddevice.one_dimensional_dots_array(
    n_dots=2)
# print the device information
qddevice.print_device_info()
```

The output generated is as follows:

Code Output

Device type: in-line array
Number of dots: 2
Number of gates: 2
Physical dot locations: $[(0,0),(1,0)]$
Dot-dot mutual capacitance matrix:

$$\begin{bmatrix} 0.12 & 0.08 \\ 0.08 & 0.12 \end{bmatrix}$$

Dot-gate mutual capacitance matrix:

$$\begin{bmatrix} 0.12 & 0.00 \\ 0.00 & 0.12 \end{bmatrix}$$

Upon specifying the `n_dots`, the `QDDevice` class autonomously initializes all requisite attributes. However, the package offers flexibility for further customization, either through standard alteration functions or by manual attribute assignment using setter methods.

The following code snippets illustrate both methods of customization.

CUSTOMIZATION VIA DEFAULT ATTRIBUTES

In this double-dot scenario, we leverage the built-in modification functions accessible through the architecture specification method. This is achieved by specifying `equal_dots = False`, `equal_gates = False`, and/or adjusting the `crosstalk_strength` parameter within a range from 0 (indicating no crosstalk) to 1 (representing the maximum threshold of crosstalk, determined in proportion to the capacitances within the simulation). For users seeking further customization, the capacitance values can be adjusted by examining and modifying the source code as necessary.

Python Code

```
from qdsim import QDDevice, QDSimulator

# create a quantum dot device object
qddevice = QDDevice()
# double dot
qddevice.one_dimensional_dots_array(
    n_dots=2, equal_dots=False,
    equal_gates=False,
    crosstalk_strength=0.3)
# print the device information
qddevice.print_device_info()
```

The output generated is as follows:

Code Output

Device type: in-line array
Number of dots: 2
Number of gates: 2
Physical dot locations: $[(0,0),(1,0)]$
Dot-dot mutual capacitance matrix:

$$\begin{bmatrix} 0.12 & 0.08 \\ 0.08 & 0.11 \end{bmatrix}$$

Dot-gate mutual capacitance matrix:

$$\begin{bmatrix} 0.13 & 0.02 \\ 0.02 & 0.15 \end{bmatrix}$$

In this example, the alteration in the dot-to-dot and dot-to-gate mutual capacitance matrices is achieved by the introduction of random values. Using `equal_dots = False` will add random values on the diagonal of the dot-to-dot mutual capacitance matrix, while `equal_gates = False` will add random values to the diagonal of the dot-to-gate mutual capacitance matrix. Setting a value for `crosstalk_strength` will add random numbers to the off-diagonal, to ensure crosstalk effects. The random values applied can be both positive and negative, and are properly scaled with respect to the order of magnitude used in the matrices to ensure the maintenance of realistic and physically plausible parameters, i.e. they would only account for small variations of the values, roughly 10-20% variations.

Conversely, for users seeking to employ custom capacitance matrices, the package provides two setter methods for this purpose: `set_dot_dot_mutual_capacitance_matrix` and `set_dot_gate_mutual_capacitance_matrix`.

Python Code

```
from qdsim import QDDevice, QDSimulator

# create a quantum dot device object
qddevice = QDDevice()

# double dot
qddevice.one_dimensional_dots_array(
    n_dots=2)

# define the custom capacitance matrices
cdd = np.array([[0.10, 0.7],[0.7, 0.15]])
cdg = np.array([[0.14, 0.3],[0.3, 0.12]])

# modify the class attributes
qddevice.
    set_dot_dot_mutual_capacitance_matrix(
        cdd)
qddevice.
    set_dot_gate_mutual_capacitance_matrix(
        cdg)

# print the device information
qddevice.print_device_info()
```

The output generated is as follows:

Code Output

Device type: in-line array
Number of dots: 2
Number of gates: 2
Physical dot locations: $[(0,0),(1,0)]$
Dot-dot mutual capacitance matrix:

$$\begin{bmatrix} 0.10 & 0.07 \\ 0.07 & 0.15 \end{bmatrix}$$

Dot-gate mutual capacitance matrix:

$$\begin{bmatrix} 0.14 & 0.03 \\ 0.03 & 0.12 \end{bmatrix}$$

When customising the capacitance matrices, it is key to pay attention to the indexing convention described in Section 5.3.1 and to guarantee the symmetry of the dot-to-dot mutual capacitance matrix.

We recommend to use `plot_device` plotting method to verify that dots and gates are ordered as intended.

The inclusion of a sensor (along with its label) in the plot for enhanced visualization, as well as the capability to export the plot to a file with a preferred format, is achieved by executing the line of code provided below.

Python Code

```
# plot the device, the sensor
# and save the plot to a file
qddevice.plot_device(
    sensor_locations=[[2,1]],
    sensor_labels=['S0'],
    save_plot_to_filepath='dqd_device.pdf')
```

It is pertinent to note that the plotting method does not render the gates. This omission is intentional for two primary reasons: firstly, the geometry of the gates is not critical for simulation objectives, as the interactions between gates and dots are encapsulated within the dot-to-gate mutual capacitance matrices. Secondly, in devices with individual control mechanisms, the indexing for gates and dots is identical. We provide a schematic depiction of gates only for the cross-bar device, where the shared control aspect (single gate controls multiple dots) requires an indexing guidance.

SIMULATION

To start the simulation of the device, create an instance of the `QDSimulator` class. This class has a method named `simulate_charge_stability_diagram`, which accepts `qd_device`

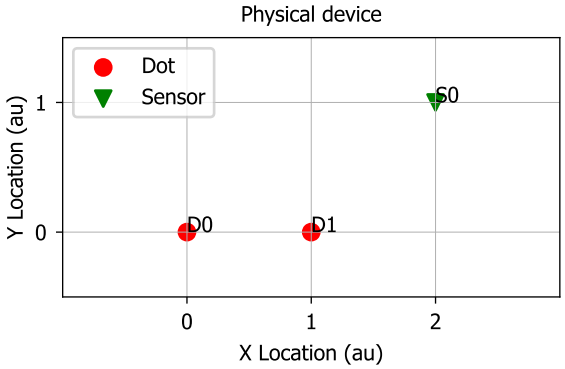


Figure 5.2: A schematic plot of the double quantum dot architecture with sensor, which is the output of the `plot_device` method.

as one of its parameters. Thus, an instance of the `QDDevice` class designated for simulation is inputted into the `QDSimulator`'s simulation method, rendering the simulator class agnostic to the specific device being simulated. Consequently, the code snippets provided herein are applicable across all use-cases discussed within this Examples section.

Python Code

```
# create a quantum dot simulator object
# simulating electrons
qdsimulator = QDSimulator(simulate=
                          'Electrons')

# set the sensor locations
qdsimulator.set_sensor_locations([[2, 1]])

# simulate the charge stability diagram
qdsimulator.
    simulate_charge_stability_diagram(
        qd_device=qddevice, solver='MOSEK',
        v_range_x=[-5, 20],
        v_range_y=[-5, 20],
        n_points_per_axis=60,
        scanning_gate_indexes=[0, 1],
        use_ray=True)
```

After initializing a `QDSimulator` object with the intention of simulating an electron-based quantum dot device, the `simulate` attribute is set to 'Electrons'. This attribute can alternatively be configured to 'Holes'. If this attribute remains unspecified, the sim-

ulation defaults to 'Electrons'.

The subsequent step is determination of the sensor locations within the simulation environment. This is achieved by specifying their coordinates in Cartesian format $[[x_0, y_0], [x_1, y_1], \ldots]$ through the `set_sensor_locations` method.

The simulation process is then executed via the `simulate_charge_stability_diagram` method, requiring the specification of several parameters. These include the `qd_device` parameter, which requires an instance of the `QDDevice` class, and the selection of an optimization solver ('MOSEK' for licensed use or 'SCIP' for an open-source option via CVXPY). When unspecified, the solver defaults to 'SCIP'. Voltage ranges along the x and y axes are defined by `v_range_x` and `v_range_y`, respectively, with `n_points_per_axis` determining the plot's resolution. The indices of the gates to be scanned are specified in `scanning_gate_indexes`, with a maximum of two gates allowed simultaneously. The indexing order is significant, as the first index always denotes the x-axis and the second the y-axis. Additionally, the attribute `use_ray=True` can be employed to leverage the Ray parallelization library [28] [29] for enhanced computational efficiency. By using Ray we can speed up the computational time by parallelizing the computation at each pixel of the plot.

PLOTTING AND ADDING NOISE

The `plot_charge_stability_diagrams` method visualizes the simulation outcomes, plotting either the potential or current landscape, with an option to incorporate noise. To visualize the sensed potential, the `plot_potential` argument should be set to `True`. Conversely, setting `plot_potential` to `False` directs the method to plot the sensed current.

For introducing noise into the visualization, three distinct types of noise can be applied: `gaussian_noise`, `white_noise`, and `pink_noise`. Activating any of these noise features is achieved by setting the corresponding attribute to `True`. The introduced noise is composed of random values added to each plot point, sourced from respective probability distributions for Gaussian and white noise, and utilizing functions from the pyplnoise [30] library for pink noise.

Following are three examples of plots for the default double quantum dot device, using the default settings:

Python Code

```
# plot the charge stability diagram

# potential, no noise
qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=True,
    gaussian_noise=False,
    white_noise=False,
    pink_noise=False)

# current, no noise
qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=False,
    gaussian_noise=False,
    white_noise=False,
    pink_noise=False)

# current, noisy
qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=False,
    gaussian_noise=True,
    white_noise=True,
    pink_noise=True)
```


The resulting plots are shown in Figure 5.3.

The noise can further adjusted through specific method attributes: `gaussian_noise_params` for setting the mean and standard deviation of the distribution, `white_noise_params` for defining the noise range, and `pink_noise_params` for adjusting the frequency and amplitude range. For instance, to configure Gaussian noise with a mean of 0.5 and a standard deviation of 0.3, one would set `gaussian_noise = True` and `gaussian_noise_params = [0.5, 0.3]`.
Lastly, the color scheme of the plot can be personalized by assigning the desired color code to the `cmapvalue` attribute.

5.4.2. THE CROSSBAR 4X4 SHARED CONTROL DEVICE

The `crossbar_array_shared_control` architecture, as presented in Ref. [7], represents another default configuration accessible to users. In contrast to the `one_dimensional_dots_array`, this architecture employs a two-dimensional grid layout for dot placement, featuring a shared control system where a single gate can simultaneously influence multiple dots. Such architectures are of particular interest within the academic community for their potential to mitigate the scalability challenge inherent to quantum dot devices. Typically, the number of gates increases linearly with the addition of dots, complicating tuning efforts for expanding device configurations. The

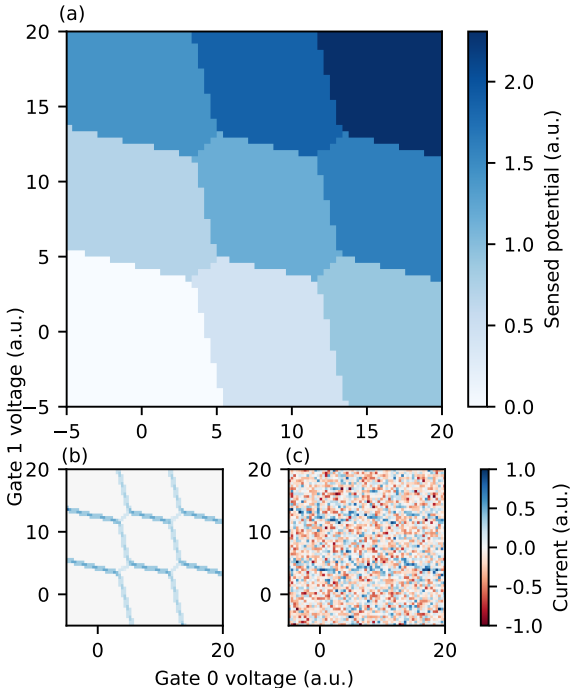


Figure 5.3: Simulated charge stability diagrams for a double dot device. In Figure (a) the potential sensed is plotted without noise. In Figure (b) and (c) the gradient is evaluated, therefore plotting the current, with and without noise.

shared control approach is considered as an option to lessen this scalability concern.

To simulate a shared-control quantum dot crossbar array, usersfirst create an instance of the `QDDevice` class and then choose the `crossbar_array_shared_control` function. Similar to the `one_dimensional_dots_array` method, specifying the `n_dots_side` argument as an integer representing the grid's side length automatically configures the device with default parameters. These parameters, may be further customized through either a selection of built-in adjustments (e.g., `equal_dots = False`, `equal_gates = False`) or via setter methods for more granular control.

In the following code box, a 4x4 shared-control quantum dot crossbar array is shown. By utilizing the built-in modification functions, we adjust the default settings, subsequently outputting the device's specifications, including the dot-to-dot and dot-to-gate mutual capacitance matrices, and visualizing the device alongside the designated sensor.

Python Code

```
# create a quantum dot device object
qddevice = QDDevice()

# crossbar array with shared control
# with 4 dots per side
qddevice.crossbar_array_shared_control(
    n_dots_side=4, equal_dots=False,
    equal_gates=False)

# print the device information
qddevice.print_device_info()

# plot device with sensors and save plot
qddevice.plot_device(
    sensor_locations=[[0,4]],
    sensor_labels=['S0'],
    save_plot_to_filepath='4x4_device.pdf')
```

Code Output

Device type: crossbar
Number of dots: 16
Number of gates: 7
Physical dot locations: [(0,3), (0,2), (1,3), (0,1), (1,2),
(2,3), (0,0), (1,1), (2,2), (3,3), (1,0), (2,1), (3,2),
(2,0), (3,1), (3,0)]
Dot-dot mutual capacitance matrix:

$$\begin{bmatrix}
0.12 & 0.08 & 0.08 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.08 & 0.12 & 0.04 & 0.08 & 0.08 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.08 & 0.04 & 0.12 & 0.00 & 0.08 & 0.08 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.08 & 0.00 & 0.12 & 0.04 & 0.00 & 0.08 & 0.08 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.04 & 0.08 & 0.08 & 0.04 & 0.12 & 0.04 & 0.00 & 0.08 & 0.08 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.08 & 0.00 & 0.04 & 0.11 & 0.00 & 0.00 & 0.08 & 0.08 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.08 & 0.00 & 0.00 & 0.12 & 0.04 & 0.00 & 0.00 & 0.08 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.04 & 0.00 & 0.08 & 0.08 & 0.00 & 0.04 & 0.12 & 0.04 & 0.00 & 0.08 & 0.08 & 0.00 & 0.04 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.04 & 0.00 & 0.08 & 0.08 & 0.00 & 0.04 & 0.12 & 0.04 & 0.00 & 0.08 & 0.08 & 0.00 & 0.04 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.08 & 0.00 & 0.00 & 0.04 & 0.12 & 0.00 & 0.00 & 0.08 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.08 & 0.08 & 0.00 & 0.00 & 0.12 & 0.04 & 0.00 & 0.08 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.08 & 0.08 & 0.00 & 0.04 & 0.11 & 0.04 & 0.08 & 0.08 & 0.04 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.08 & 0.08 & 0.00 & 0.04 & 0.11 & 0.00 & 0.08 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.08 & 0.08 & 0.00 & 0.11 & 0.04 & 0.08 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.00 & 0.08 & 0.08 & 0.04 & 0.12 & 0.08 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.08 & 0.08 & 0.12
\end{bmatrix}$$

Dot-gate mutual capacitance matrix:

$$\begin{bmatrix}
0.15 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.18 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.18 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.12 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.12 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.12 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.13 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.13 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.13 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.13 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.13 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.13 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.13 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.16 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.16 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.13
\end{bmatrix}$$

In this architecture, gates are arranged diagonally, with the indexing of both dots and gates illustrated in Figure 5.4.

SIMULATION

Similar to the double dot device scenario, simulating the device requires an instance of the `QDSimulator` class. While it's not imperative to create a new simulator instance for each device instance, allowing for the reuse of a single simulator instance, there are

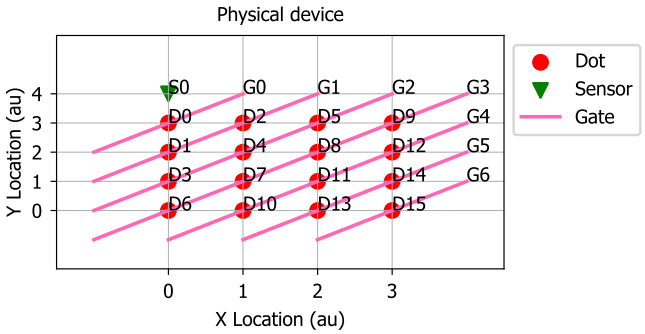


Figure 5.4: A schematic plot of the crossbar 4x4 shared control architecture with sensor, which is the output of the `plot_device` method.

situations where dedicating a separate simulator instance to each device may facilitate direct comparisons between simulations. The choice of approach is left to the user's discretion based on their specific requirements.

Following the selection of the physical phenomena to be simulated (either electrons or holes), the user employs the `set_sensor_locations` method to specify the Cartesian coordinates of the sensor(s) involved in the simulation. Subsequently, the `simulate_charge_stability_diagram` function is executed. This involves selecting the device, selecting a solver, setting the voltage ranges, determining the simulation's resolution (noting that higher resolution increases computational demand), and identifying the gates to be scanned.

This example differs from the prior shown in Section 5.4.1 due to the presence of additional gates beyond those being actively scanned. To ensure a successful simulation, it is necessary to define the voltage settings for these additional gates. There are two approaches to manage gate voltages in scenarios with more than two gates: uniformly applying voltages using `fixed_voltage` or customizing voltages for each gate via `gates_voltages`. For instance, in a setup with three gates, where Gates 0 and 2 are being scanned and Gate 1 is set to 1.5 volts, this would be represented as `gates_voltages = [None, 1.5, None]`. Alternatively, the same outcome would be achieved by setting `fixed_voltage = 1.5`.

In scenarios with more than three gates, employing `fixed_voltage = 1.5` assigns a uniform voltage of 1.5 to all gates not under scan. In contrast, the `gates_voltages` option permits users to selectively assign specific voltage values to each of the un-scanned gates as per their preference.

It is required to exclusively use either `gates_voltages` or `fixed_voltage` for specifying gate voltages.

In the following code snippet, we assign a uniform voltage of 1 to all gates of secondary interest (specifically Gates 2, 3, 4, 5, and 6).

Python Code

```
# create a quantum dot simulator object
# simulating electrons
qdsimulator = QDSimulator(simulate=
                          'Electrons')

# set the same sensor locations
qdsimulator.set_sensor_locations([[0, 4]])

# simulate the charge stability diagram
qdsimulator.
    simulate_charge_stability_diagram(
        qd_device=qddevice, solver='MOSEK',
        v_range_x=[-5, 20],
        v_range_y=[-5, 20],
        n_points_per_axis=60,
        scanning_gate_indexes=[0, 1],
        fixed_voltage=1, use_ray=True)
```


Employing the identical functions used in the double dot scenario to plot the simulated charge stability diagrams, we generate the subsequent plots shown in Figure 5.5.

Python Code

```
# plot the charge stability diagram

# potential, no noise
qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=True,
    gaussian_noise=False,
    white_noise=False,
    pink_noise=False)

# current, no noise
qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=False,
    gaussian_noise=False,
    white_noise=False,
    pink_noise=False)

# current, noisy
qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=False,
    gaussian_noise=True,
    white_noise=True,
    pink_noise=True)
```


GETTING THE CHARGE CONFIGURATION

In order to simplify the labelling of the polytopes, users can take advantage of the `get_charge_configuration` method of the `QDSimulator` class. Users can provide in input a tuple representing the voltage coordinates in the charge stability diagram, and the method will automatically evaluate the closest simulated point and provide the charge configuration at that point. The need to approximate the voltage point comes from the limitation in the granularity of the plot.

An example on how to use the function is shown below.

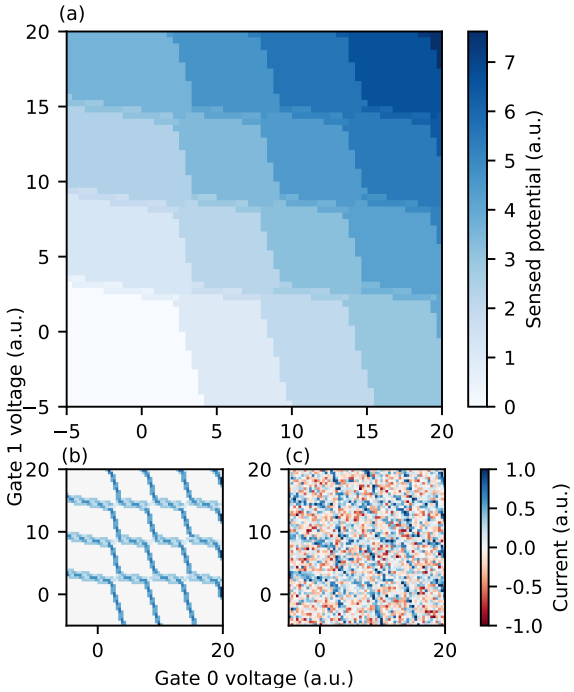


Figure 5.5: Simulated charge stability diagrams for a 4x4 shared-control dot device. In Figure (a) the potential sensed is plotted without noise. In Figure (b) and (c) the gradient is evaluated, therefore plotting the current, with and without noise correspondingly.

Python Code

```
# Let's test the empty region
voltage_point = [0, 0]

# Get the charge configuration of a
# point in the charge stability diagram

print('Charge configuration at chosen
    point', voltage_point, ':')
qdsimulator.get_charge_configuration(
    voltage_point=voltage_point)
```

Code Output

```
Charge configuration at chosen point [0, 0] :
Voltage point considered: [0.08474576 0.08474576 1. 1. 1. 1. 1. ]
Charge configuration: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

It is also possible to save the voltage point-charge configuration couple into a variable, as in the following code:

Python Code

```
# Let's test another region
voltage_point = [7, 0]

# Get the charge configuration of a
# point in the charge stability diagram

print('Charge configuration at chosen
    point', voltage_point, ':')
voltage_and_charge_config =
    qdsimulator.get_charge_configuration(
        voltage_point=voltage_point)


print("voltage_and_charge_config =",
    voltage_and_charge_config)
```

Code Output

```
Charge configuration at chosen point [7, 0] :
Voltage point considered: [6.86440678 0.08474576 1. 1. 1. 1. 1. ]
Charge configuration: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
voltage_and_charge_config =
(array([6.86440678, 0.08474576, 1. , 1. , 1. , 1. , 1. ]), array([1., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 0.]))
```

5.4.3. CUSTOM DEVICE CONFIGURATION

Users have two main options for configuring custom devices. The first option involves only specifying the locations of the quantum dots. In this case, an individual control system is assumed (one gate per dot), and standard mutual capacitance matrices are derived based on a simple model that considers distance, focusing on first and second nearest neighbors interactions. The second option allows for the further manual specification of the mutual capacitance matrices between dots (dot-to-dot) and between dots and gates (dot-to-gate), thus enabling simulation of complex gate-dot configurations.

Below we show examples of both approaches to custom device simulation.

INDIVIDUAL CONTROL: DOT LOCATION SPECIFICATION

First, we consider a scenario where the simulation of a custom device focuses solely on the placement of quantum dots. This can be achieved by creating an instance of the `QDDevice` class and utilizing the `set_custom_dot_locations` method. The order of coordinates tuples directly maps to the dots' indices. The use of `set_custom_dot_locations`

triggers an internal function that calculates the mutual capacitance matrices for both dot-to-dot and dot-to-gate interactions. As a result, printing the device information shows these matrices as configured attributes within the class. Similar to the method for the default architectures shown above, minor customizations can be implemented through the boolean parameters `equal_dots`, `equal_gates`, and `crosstalk_strength` for adjusting crosstalk strength. Additionally, the default capacitance value, set at 0.12, can be altered via the `c0` parameter to suit specific requirements.

Python Code

```
# create a quantum dot device object
qddevice = QDDevice()

# set the custom dot locations
qddevice.set_custom_dot_locations([[2, 2],
    [3, 1.5], [4, 2], [1, 1], [5, 1],
     [2, 0], [3, 0.5], [4, 0]],
    equal_dots=False, equal_gates=False,
    crosstalk_strength=0.2, c0=0.12)

# print the device information
qddevice.print_device_info()

# plot the device with sensor
qddevice.plot_device(
    sensor_locations=[[1,2]],
    sensor_labels=['S0'])
```

Code Output

Device type: custom
Number of dots: 8
Number of gates: 8
Physical dot locations: [[2, 2], [3, 1.5], [4, 2], [1, 1], [5, 1], [2, 0], [3, 0.5], [4, 0]]
Dot-dot mutual capacitance matrix:

$$\begin{bmatrix} 0.12 & 0.06 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.06 & 0.12 & 0.06 & 0.00 & 0.00 & 0.00 & 0.08 & 0.00 \\ 0.00 & 0.06 & 0.12 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 \\ 0.04 & 0.00 & 0.00 & 0.13 & 0.00 & 0.04 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.04 & 0.00 & 0.11 & 0.00 & 0.00 & 0.04 \\ 0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.12 & 0.06 & 0.00 \\ 0.00 & 0.08 & 0.00 & 0.00 & 0.00 & 0.06 & 0.13 & 0.06 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.06 & 0.12 \end{bmatrix}$$

Dot-gate mutual capacitance matrix:

$$\begin{bmatrix} 0.14 & 0.01 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.01 & 0.13 & 0.01 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 \\ 0.00 & 0.01 & 0.12 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 \\ 0.01 & 0.00 & 0.00 & 0.12 & 0.00 & 0.01 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.01 & 0.00 & 0.10 & 0.00 & 0.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.10 & 0.02 & 0.00 \\ 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.02 & 0.14 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.01 & 0.13 \end{bmatrix}$$

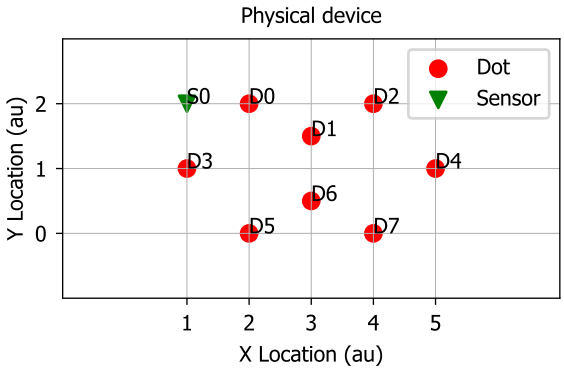


Figure 5.6: A schematic plot of the custom design with sensor, which is the output of the `plot_device` method.

Figure 5.6 shows an outline of the custom device, allowing verification of the dot indices against the provided list of coordinates. Currently, a schematic depiction of the

gates in custom designs is not available, though such a feature may be incorporated in future releases. In this case each dot is controlled by an individual gate. For the device simulation itself we simply repeat the procedure defined for the default devices: create a simulator instance, specify the sensor(s) locations, and execute the `simulate_charge_stability_diagram` method. Note that, despite the system featuring more than two gates, neither the `fixed_voltage` nor the `gate_voltages` parameters are employed. This is due to reliance on the default setting where, in the absence of explicit specifications for these parameters, the system defaults to `fixed_voltage = 0`.

The outcome of the simulation can be plotted like so (output shown in Figure 5.7):

Python Code

```
# create a quantum dot simulator object
qdsimulator = QDSimulator(simulate=
                          'Electrons')

# set the sensor locations
qdsimulator.set_sensor_locations([[1, 2]])

# simulate the charge stability diagram
qdsimulator.
    simulate_charge_stability_diagram(
        qd_device=qddevice, solver='MOSEK',
        v_range_x=[-5, 20],
        v_range_y=[-5, 20],
        n_points_per_axis=60,
        scanning_gate_indexes=[0, 3],
        use_ray=True)

# plot the charge stability diagrams
qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=True,
    gaussian_noise=False, white_noise=False,
    pink_noise=False)

qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=False,
    gaussian_noise=False, white_noise=False,
    pink_noise=False)

qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=False,
    gaussian_noise=True, white_noise=True,
    pink_noise=True)
```

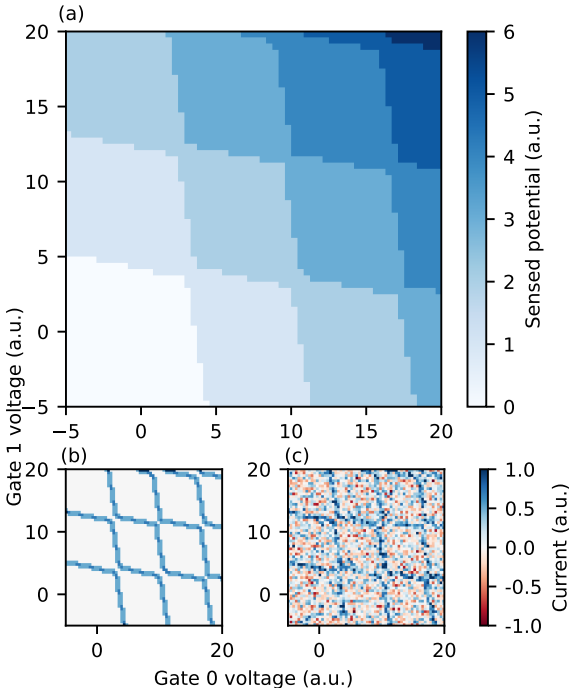


Figure 5.7: Simulated charge stability diagrams for a custom design with individual control. In Figure (a) the potential sensed is plotted without noise. In Figure (b) and (c) the gradient is evaluated, therefore plotting the current, respectively with and without noise.

SHARED CONTROL: SPECIFYING THE DOT-TO-GATE MUTUAL CAPACITANCE MATRIX

In this example, we evolve the previously discussed custom model by transitioning from an individual control system to introducing a custom shared control system, reflected through modifications to the dot-to-gate mutual capacitance matrix.

We adopt a notation that allows us to specify which dots are controlled by which gate. Specifically, let's assume we want to use four gates overall to control all dots. We add one of the following four letters: n, w, e, s, representing north, west, east, and south, respectively. Dots sharing the same letter are understood to be under the control of the same gate.

Let us illustrate this concept on the concrete example:

Python Code

```
# create a quantum dot device object
qddevice = QDDevice()

# set the custom dot locations
qddevice.set_custom_dot_locations([[2, 2],
    [3, 1.5], [4, 2], [1, 1], [5, 1],
    [2, 0], [3, 0.5], [4, 0]],
    equal_dots=False, equal_gates=False,
    crosstalk_strength=0.2, c0=0.12)

# plot the device with custom labels
# and sensor
qddevice.plot_device(
    sensor_locations=[[1,2]],
    sensor_labels=['S0'],
    custom_dot_labels=['D0n', 'D1n', 'D2n',
        'D3w','D4e', 'D5s','D6s', 'D7s'])
```

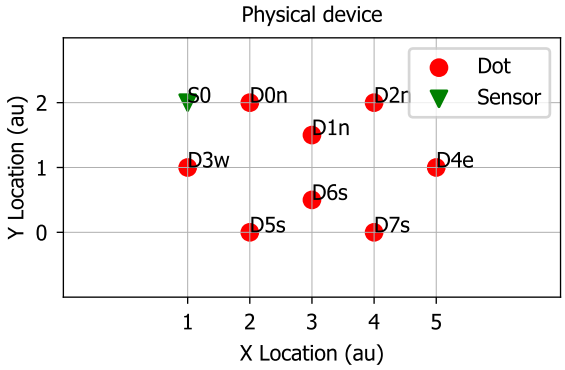



Figure 5.8: Another schematic plot of the custom design, in which the labels of the dots have been manually changed to adhere to the new labelling system: 'D' stands for dot, the integer number represents the dot index, the letter represents the gate controlling the dot.

In the example above, Dots 0, 1, and 2 are controlled by the northern gate, Dot 3 is governed by the western gate, Dot 4 responds to the eastern gate, while Dots 5, 6, and 7 are controlled by southern gate.

The next step involves encoding this gate-dot architecture into the dot-to-gate mutual capacitance matrix. The dot-to-dot mutual capacitance matrix remains unchanged, still calculated automatically based on the proximity of the dots. For the purpose of matrix notation, we adopt the indexing scheme $[n:0, w:1, e:2, s:3]$, needed for the construction of an 8x4 matrix representing the dot-to-gate interactions.

This conversion is accomplished through the code snippet below, wherein we employ the `set_dot_gate_mutual_capacitance_matrix` setter method. This allows us to update the dot-to-gate mutual capacitance matrix from the default, automatically generated one, indicative of individual control, to our newly conceptualized matrix that reflects the shared control system.

Python Code

```
dot_gate_matrix = np.array([
    [0.14, 0.00, 0.00, 0.00],
    [0.14, 0.00, 0.00, 0.00],
    [0.14, 0.00, 0.00, 0.00],
    [0.00, 0.13, 0.00, 0.00],
    [0.00, 0.00, 0.12, 0.00],
    [0.00, 0.00, 0.00, 0.15],
    [0.00, 0.00, 0.00, 0.15],
    [0.00, 0.00, 0.00, 0.15]
])

qddevice.
    set_dot_gate_mutual_capacitance_matrix(
        dot_gate_matrix)

qddevice.print_device_info()
```

Code Output

Device type: custom
Number of dots: 8
Number of gates: 4
Physical dot locations: [[2,2],[3,1.5],[4,2],[1,1],[5,1],[2,0],[3,0.5],[4,0]]
Dot-dot mutual capacitance matrix:

$$\begin{bmatrix} 0.12 & 0.06 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.06 & 0.12 & 0.06 & 0.00 & 0.00 & 0.00 & 0.08 & 0.00 \\ 0.00 & 0.06 & 0.12 & 0.00 & 0.04 & 0.00 & 0.00 & 0.00 \\ 0.04 & 0.00 & 0.00 & 0.12 & 0.00 & 0.04 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.04 & 0.00 & 0.12 & 0.00 & 0.00 & 0.04 \\ 0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.12 & 0.06 & 0.00 \\ 0.00 & 0.08 & 0.00 & 0.00 & 0.00 & 0.06 & 0.12 & 0.06 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.04 & 0.00 & 0.06 & 0.12 \end{bmatrix}$$

Dot-gate mutual capacitance matrix:

$$\begin{bmatrix} 0.14 & 0.00 & 0.00 & 0.00 \\ 0.14 & 0.00 & 0.00 & 0.00 \\ 0.14 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.13 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.12 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.15 \\ 0.00 & 0.00 & 0.00 & 0.15 \\ 0.00 & 0.00 & 0.00 & 0.15 \end{bmatrix}$$

Finally, we proceed to simulate and plot the charge stability diagrams in Figure 5.9.

Python Code

```
# create a quantum dot simulator object
qdsimulator = QDSimulator(simulate=
                              'Electrons')

# set the sensor locations
qdsimulator.set_sensor_locations([[2, 1]])

# simulate the charge stability diagram
qdsimulator.
    simulate_charge_stability_diagram(
        qd_device=qddevice, solver='MOSEK',
        v_range_x=[-5, 20],
        v_range_y=[-5, 20],
        n_points_per_axis=60,
        scanning_gate_indexes=[0, 3],
        use_ray=True)

# plot the charge stability diagrams
qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu',plot_potential=True,
    gaussian_noise=False, white_noise=False,
    pink_noise=False)

qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu',plot_potential=False,
    gaussian_noise=False, white_noise=False,
    pink_noise=False)

qdsimulator.plot_charge_stability_diagrams(
    cmapvalue='RdBu', plot_potential=False,
    gaussian_noise=True, white_noise=True,
    pink_noise=True)
```

5.5. PERFORMANCE EVALUATION AND CONSTRAINTS

QDsim package is designed to equip the scientific community with a user-friendly tool for simulating charge stability diagrams of arbitrary architectures. Our primary goal is to create an accessible codebase, allowing researchers to efficiently simulate charge stability diagrams for extensive quantum dot arrays with a minimal setup and little preliminary knowledge. The ultimate objective of QDsim package is to introduce a rapid and efficient tool for generating comprehensive datasets for subsequent use in machine learning model training.

Among comparable available tools, the QTT package [31] stands out, although it pri-

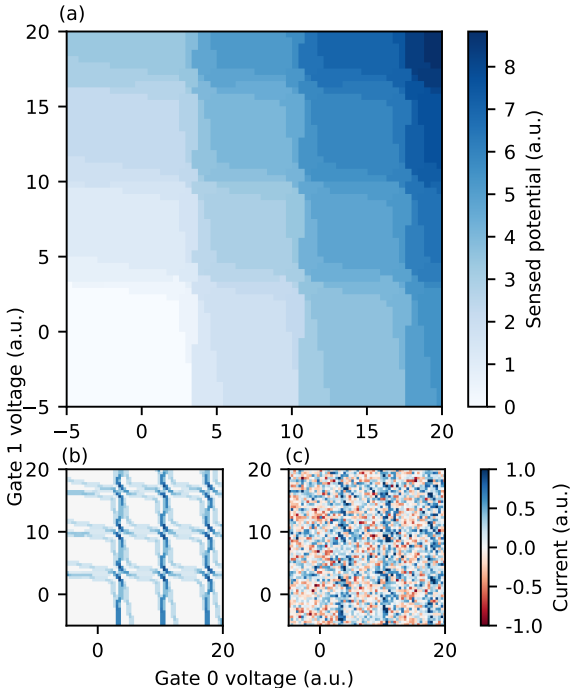


Figure 5.9: Simulated charge stability diagrams for a custom design with shared control. In Figure (a) the potential sensed is plotted without noise. In Figure (b) and (c) the gradient is evaluated, therefore plotting the current, respectively with and without noise.

marily addresses highly specialized experimental scenarios. `QTT` is geared more towards analysis and measurements within precise experimental frameworks, rendering it somewhat challenging for beginners and limiting its applicability to a narrow range of devices (e.g., double dots, in-line 6 dot array, triple dot, square dot) all under individual control schemes. This specificity in focus is the primary reason a detailed comparison with `QTT` has not been pursued, as the two packages cater to distinct needs and applications. Furthermore, very recently a new package named `SimCATS` [32] has been published. It aims to achieve a maximally realistic description of CSD especially with regards to sensing and noise implementations. While having a different focus, it caters to the need of realistic noise and could be a promising addition to our approximated solution. Furthermore, our work is complementary to a simultaneously published package called `QDarts` [33], which leverages a polytope-finding algorithm to efficiently simulate and locate charge transitions in the presence of tunnel couplings, non-constant charging energy and realistic noise. However due higher computational complexity, it focuses on smaller dot arrays (approximately 10 dots).

Focusing on the strengths of our package, `QDsim` distinguishes itself with its speed and capability to simulate extensive arrays. To quantify these advantages, all simulations and time measurements discussed herein were conducted on an Apple M2 chip equipped with 16 GB of memory.

Speed optimization in the package is influenced by two main factors: the device architecture size (i.e. number of dots and gates), reflected by the matrix dimensions, which offers limited parallelization opportunities, and the plot granularity, which can benefit from advanced parallelization techniques.

Here we compare across several configurations, including double dots, in-line 6 dots, and shared control arrays of sizes 4x4, 6x6, and 8x8.

The comparison results are illustrated in Figure 5.10.

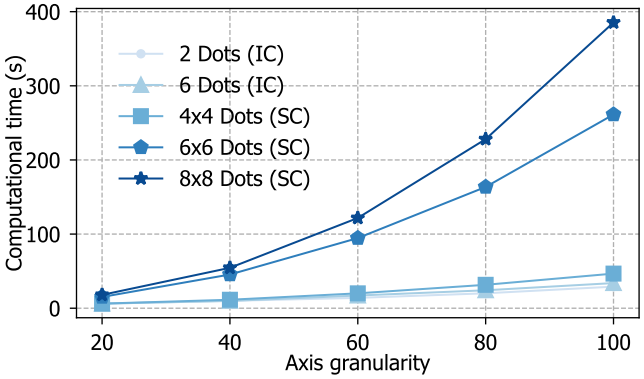


Figure 5.10: Comparing the computational time (in seconds) to simulate a specific architecture, with respect to the granularity of the plot's axis. The number of points evaluated for the plot is therefore the x-axis numbers squared. From the plot we can see how the computational time increases with increasing number of plot points, but also with the size of the device, i.e. more dots require more computational time for a given axis granularity. A 64-dot device with shared control system can be simulated with good granularity on a laptop in less than 4 minutes.

Granularity requirements will vary based on the desired plot quality and voltage space size; naturally, broader voltage ranges would likely necessitate higher granularity levels. For instance, in our example section, a voltage range of 25 and a granularity level of 60 yielded satisfactory plots in under a minute, highlighting the package's emphasis on speed and ease of use.

When considering limitations, we identify two main categories: physical and functional constraints. On the physical side, it is crucial to acknowledge that the simulations are based on electrostatics and do not account for quantum mechanical effects. This limitation impacts the simulations' realism, potentially affecting the precision in modeling actual devices. However, the package's primary goal is not to perfectly replicate physical systems but rather to generate datasets for machine learning applications, where models can be initially trained on sufficiently accurate simulated data and later fine-tuned with real experimental data.

From a functional perspective, potential enhancements could include improved visualizations of gate configurations, optimization of parallel processing routines for increased speed, and the development of a graphical user interface (GUI). This GUI could allow users to intuitively position dots and gates, with the software automatically suggesting starting points for mutual capacitance values between dots and gates.

5.6. CONCLUSION

The development of this package was driven by the ambition to offer the scientific community a highly accessible and user-friendly tool, specifically designed to streamline the generation of charge stability diagrams for extensive quantum dot arrays. With a focus on efficiency and speed, this package aims to significantly reduce the time and complexity traditionally associated with such simulations. By simplifying the process for both students and professionals, whether in theoretical or experimental domains, the package opens up new avenues for exploration and discovery in the field of quantum computing and nanotechnology. Furthermore, it lays the groundwork for the creation of comprehensive datasets, essential for the advancement of machine learning applications within this sphere.

As we look forward to contributions from the community, enriching the package with more sophisticated models, our ultimate hope is that it becomes a cornerstone for innovation, fostering advancements that leverage both computational simulations and machine learning to unravel the complexities of quantum systems.

REFERENCES

[1] Valentina Gualtieri et al. “QDsim: A user-friendly toolbox for simulating large-scale quantum dot devices”. In: *SciPost Physics Codebases* (2025), p. 046.

[2] Guido Burkard et al. “Semiconductor spin qubits”. In: *Reviews of Modern Physics* 95.2 (2023), p. 025003.

[3] Daniel Loss and David P DiVincenzo. “Quantum computation with quantum dots”. In: *Physical Review A* 57.1 (1998), p. 120.

[4] Anasua Chatterjee et al. “Semiconductor qubits in practice”. In: *Nature Reviews Physics* 3.3 (2021), pp. 157–177.

[5] LMK Vandersypen et al. “Interfacing spin qubits in quantum dots and donors—hot, dense, and coherent”. In: *npj Quantum Information* 3.1 (2017), p. 34.

[6] Stephan GJ Philips et al. “Universal control of a six-qubit quantum processor in silicon”. In: *Nature* 609.7929 (2022), pp. 919–924.

[7] Francesco Borsoi et al. “Shared control of a 16 semiconductor quantum dot crossbar array”. In: *Nature Nanotechnology* 19.1 (2024), pp. 21–27.

[8] Renato Durrer et al. “Automated tuning of double quantum dots into specific charge states using neural networks”. In: *Physical Review Applied* 13.5 (2020), p. 054019.

[9] Sandesh S Kalantre et al. “Machine learning techniques for state recognition and auto-tuning in quantum dots”. In: *npj Quantum Information* 5.1 (2019), p. 6.

[10] Rouven Koch et al. “Adversarial Hamiltonian learning of quantum dots in a minimal Kitaev chain”. In: *Physical Review Applied* 20.4 (2023), p. 044081.

[11] Dominic T Lennon et al. “Efficiently measuring a quantum device using machine learning”. In: *npj Quantum Information* 5.1 (2019), p. 79.

[12] Justyna P Zwolak et al. “Autotuning of double-dot devices in situ with machine learning”. In: *Physical review applied* 13.3 (2020), p. 034075.

[13] Jonas Schuff et al. “Fully autonomous tuning of a spin qubit”. In: *arXiv preprint arXiv:2402.03931* (2024).

[14] Jozef Bucko et al. “Automated reconstruction of bound states in bilayer graphene quantum dots”. In: *Physical Review Applied* 19.2 (2023), p. 024015.

[15] Stefanie Czischek et al. “Miniaturizing neural networks for charge state autotuning in quantum dots”. In: *Machine Learning: Science and Technology* 3.1 (2021), p. 015001.

[16] Jana Darulová et al. “Autonomous tuning and charge-state detection of gate-defined quantum dots”. In: *Physical Review Applied* 13.5 (2020), p. 054005.

[17] V Nguyen et al. “Deep reinforcement learning for efficient measurement of quantum devices”. In: *npj Quantum Information* 7.1 (2021), p. 100.

[18] Justyna P Zwolak et al. “Data Needs and Challenges of Quantum Dot Devices Automation: Workshop Report”. In: *arXiv preprint arXiv:2312.14322* (2023).

[19] Leo P Kouwenhoven et al. “Electron transport in quantum dots”. In: *Mesoscopic electron transport* (1997), pp. 105–214.

[20] Shuo Yang, Xin Wang, and S Das Sarma. “Generic Hubbard model description of semiconductor quantum-dot spin qubits”. In: *Physical Review B* 83.16 (2011), p. 161301.

[21] Thomas Ihn. *Semiconductor Nanostructures: Quantum states and electronic transport*. OUP Oxford, 2009.

[22] James Clerk Maxwell. *A treatise on electricity and magnetism*. Vol. 1. Oxford: Clarendon Press, 1873.

[23] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[24] MI Gil’. “On invertibility and positive invertibility of matrices”. In: *Linear Algebra and its Applications* 327.1-3 (2001), pp. 95–104.

[25] Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge, UK ; New York: Cambridge University Press, 2004. ISBN: 978-0-521-83378-3.

[26] Charles Renshaw-Whitman. “Electrostatic Modelling of Quantum Dot Arrays”. Master’s thesis. Delft, Netherlands: Delft University of Technology, 2022.

[27] Valentina Gualtieri et al. *QDsim*. https://gitlab.com/QMAI/papers/qdsim/. Version 0.0.1. https://gitlab.com/QMAI/papers/qdsim/, Mar. 2024. URL: https://gitlab.com/QMAI/papers/qdsim/.

[28] Philipp Moritz et al. “Ray: A distributed framework for emerging {AI} applications”. In: *13th USENIX symposium on operating systems design and implementation (OSDI 18)*. 2018, pp. 561–577.

[29] Ray Project Contributors. *Ray: a unified framework for scaling AI and Python applications*. https://github.com/ray-project/ray. Version 2.7.1. 2023.

[30] Jan Waldmann. *pyplnoise: Arbitrarily long streams of power law noise using NumPy and SciPy*. https://github.com/janwaldmann/pyplnoise/. Version 1.4. 2022.

[31] QuTech-Delft Contributors. *QTT*. https://github.com/QuTech-Delft/qtt/. 2023.

[32] Fabian Hader et al. “Simulation of Charge Stability Diagrams for Automated Tuning Solutions (SimCATS)”. In: *TechRxiv* (2024).

[33] Jan Krzywda et al. *QDarts*. Version 1.0.1. https://github.com/condensedAI/QDarts, Apr. 2024. DOI: 10.5281/zenodo.1234567.

6

Machine Learning for Quantum Dot Experiments: From Dataset Curation to Measurement Completion

Quantum dots offer significant potential for quantum computing, but tuning their charge states by measuring charge stability diagrams (CSDs) is both time-intensive and resource-demanding. While measurement techniques have improved in speed and signal-to-noise ratio, the inherent variability in experimental conditions often leads to the acquisition of low-quality or noisy CSDs, hindering the development of robust automated tuning algorithms. To address this, we first explore a supervised machine learning approach to classify CSDs based on their quality, thereby enabling the construction of high-quality experimental datasets. By fine-tuning a ResNet18-based classifier on a small, human-labeled subset of CSDs, we demonstrate the ability to rapidly curate large datasets of CSDs from a large number of experimental measurements. This curated dataset is then leveraged to train and evaluate generative models. Specifically, we investigate the use of diffusion models to reconstruct CSDs from partial measurements, aiming to reduce experimental time.

The results of this chapter are part of the manuscript: **V. Hernandes**, J. Rogers, T. Spriggs, R. Koch, E. Greplova, "Diffusion Model Reconstruction of Charge Stability Diagrams for Quantum Dots", in preparation (2025), and the dataset: S. de Snoo, **V. Hernandes**, T. Spriggs, et al., "Delft Charge Stability Diagram Dataset", Zenodo (2025) [1]

6.1. INTRODUCTION

Quantum computing has the potential to impact several fields, including cryptography, optimization, and materials discovery. To realize this potential, it is essential to identify scalable and practical qubit implementations. Quantum dots stand out as a modern platform for quantum computing, thanks to their scalability, long coherence times, and compatibility with semiconductor fabrication techniques [2, 3, 4, 5, 6, 7]. Quantum dots are typically implemented in 2D quantum well heterostructures. By carefully designing gate voltages, individual electrons can be isolated and controlled, forming charge or spin qubits. Adjacent dots can interact to enable two-qubit gates.

Despite their potential, operating quantum dots reliably requires extensive tuning. Standard procedures involve mapping electron configurations as a function of gate voltages, forming a charge stability diagram (CSD) [8]. These diagrams offer a visual representation of charge states, with boundaries indicating a transition to a different state. Generating CSDs requires high-resolution measurements across a multidimensional parameter space, which is both time-consuming and resource-intensive. Traditional approaches rely on DC measurements, which, although accurate, are inherently slow. RF reflectometry greatly accelerates this process, allowing tuning quantum dots in video-mode. Nonetheless, further advances will be needed to effectively automate the tuning of thousands or millions of quantum dots in future quantum chips [9, 10].

As the number of quantum dots and gate voltages to be controlled increases, manual tuning becomes impractical. Automation of this process is essential, and numerous machine learning approaches have been implemented to automate the tuning routine [11, 12, 13, 14, 15, 16, 17, 18, 19]. Existing works have focused on discriminative techniques, which use existing data to perform some prediction or classification. More recently, generative models, which aim to create new data or reconstruct missing information, have been extensively used in all domains, but their application in quantum dots tuning remains largely unexplored. The ability to generate new data based on partial measurements could be leveraged to reduce the amount of experimental data required for tuning quantum dots [20].

A fundamental challenge in applying machine learning techniques to experimental routines in quantum dot systems is the inherent noise and diversity present in the data. Experimental charge stability diagrams can vary significantly depending on factors like environmental noise, device imperfection, or even the measurement technique: radio-frequency reflectometry, for example, can return the same CSD with different noise profiles depending on the integration time chosen [21]. In order to build robust machine learning based routines that work reliably on experimental data, the availability of high-quality datasets is crucial [22]. During tuning routines of quantum dots, a large quantity of data is produced, however, not all of this raw data is beneficial to be included in a dataset used to train machine learning models, due to the presence of a vast amount of noisy or non-representative samples. Therefore, the ability to systematically curate and filter experimental data to obtain high-quality datasets is critical for the development of automated tuning algorithms. While one could, in principle, manually label every obtained CSD to build a high-quality dataset, this approach is prohibitively time consuming and labor intensive, especially considering the large amount of data needed to train accurate machine learning models [23]. To overcome this bottleneck, a powerful

technique consists on using an ML model trained of a small manually labeled dataset, used to automatically classify (and therefore label) a larger raw experimental dataset [24, 25].

In this work, we first address the challenge of building high-quality experimental quantum dot measurement datasets, by classifying CSDs based on their visual quality. We leverage representation learning, by fine-tuning ResNet18 models to perform binary classification, trained on a small carefully human-labeled subset of a large CSD dataset. This allows us to efficiently build controllable high-quality CSD dataset that can be used to train subsequent machine learning models tailored to tasks like automated tuning routines.

Following the dataset curation, we show preliminary results on how the high-quality dataset can be used to train diffusion models [26, 27] for reconstructing partially measured CSDs, reducing the amount of data that needs to be collected. We propose two measurement protocols based on grid-like voltage slices, where the diffusion model completes the missing data. This marks the first application of diffusion models in experimental quantum dot systems, opening up new possibilities for efficient and scalable quantum dot tuning.

The following of the chapter consists of methods first regarding the approach of building a high-quality CSD dataset by using a model fine-tuned on a small human-labeled dataset, and then about the diffusion model application of inpainting CSDs to reconstruct measurement data. Then, results are shown following the same structure, starting with the CSD classifier, followed by the diffusion model. We finalize the chapter with a conclusion revising the main finding and open challenges.

6.2. CSD AUTOMATED LABELING

6.2.1. CSD DATASET ACQUISITION AND CURATION

Our research begins with a large collection of approximately 120 thousand raw charge stability diagrams measurements. Each measurement consists of a **h5**-structured file with information about voltages and respective measured signal. These measurements were obtained over a long period of time from different quantum dot devices. This inherent heterogeneity in the raw dataset means that image quality varies significantly, encompassing highly resolved diagrams, noisy measurements, and incomplete or uninterpretable patterns. Figure 1 shows an illustration showing the data composition, labeled by the name of the device from which the sample comes from, and examples of significant CSDs for selected devices.

As a first step, all files are used to build a CSD **image dataset**, by simply plotting all the signals as function of the voltages, choosing image transformations and rescaling techniques so that all figures have consistent size and color.

Then, to establish a gold standard for CSD quality, a crucial step was human labeling. A small representative subset of approximately 4000 CSDs images was randomly selected from the large *image dataset*, taking into account origin device so to have a balanced dataset. The images were split into 4 subsets of approximately 1,000 samples, each of which has been manually labeled by four experienced researched. The labels assigned were **clean**, **intermediate**, and **noisy**, reflecting the utility of the CSD for subsequent

tasks, based on the noise and features present. The labeling task was carried out by using a custom GUI, that loads a CSD, give labeling options, and based on the label chosen, saves the pair CSD-label in a text file.

6.2.2. Ensemble Classifier for CSD Quality Assessment

We formulated the task of labeling CSDs as a binary classification problem: disgtinguishing between "clean/intermediate" and "noisy" CSDs. Given the imbalance present in the dataset obtained from manual labeling, and since our goal is to construct an ensemble of classifiers, we constructed five balanced training datasets, each containing 540 "clean/intermediate" CSDs, this number being all the CSDs classified as clean or intermediate by the labelers, and 540 "noisy" CSDs, randomly sampled from the 3523 CSDs classified as "noisy" by the labelers. This balancing strategy mitigates the risk of the classifier becoming biased towards the majority class during training. Each of this balanced datasets was randomly split into a training set (85%) and a validation set (15%), to track model performance during training, using batch size equal to 64.

In order to maximally leverage representation learning power, instead of training networks from scratch to classify the CSDs, we fine-tuned pre-trained ResNet18 [28]. A fully connected layer was added to each pre-trained model, connecting the number of features output by ResNet18 to one output unit, so that the networks can perform binary classification (0 or 1 being "noisy" and "clean/intermediate" CSDs, respectively). Training was performed by freezing the weights of the original model, and only updating the last layer weights, for 5 epochs, using Adam optimizer with learning rate set to 3×10^{-4}, and Binary Cross-Entropy as loss function. Five models were fine-tuned using the five different training sets described in the previous paragraph, with a balanced number of CSD per class.

The trained ensemble was then used to classify the larger, noisy, raw dataset, by employing a simple voting mechanism. For each unlabeled CSD image, all five trained models predict a logit score, from 0 to 1, representing their confidence in the image belonging to the "clean/intermediate" class. An image was assigned the label "clean" if at least three out of the five networks produce a logit score exceeding a predefined threshold. Specifically, we used threshold of **0.3**, **0.5**, **0.8**, and **0.95**, expecting that a higher threshold would imply CSDs with low noise and well defined CSD image-like features, such as Coulomb diamonds and color gradients.

6.3. Diffusion Models for Inpainting CSDs

Diffusion models (DMs) are a class of generative models that iteratively transform random noise into structured data through a process of denoising [27]. Differently from other generative models, like variational auto-encoders or generative adversarial networks, DMs to do generate data directly from a latent space, but instead are trained to reverse a gradual diffusion process, where noise is gradually added to the data until it becomes indistinguishable from Gaussian noise.

The forward diffusion process starts with a clean data sample x_0, in our case a charge stability diagram. At each step t, a small amount of Gaussian noise is added to the sample x_{t-1} to get x_t, and the process is repeated T times, until the final image x_T consists

of pure noise. An example of the forward diffusion process applied to a CSD is shown in Figure.

The goal of the DM is to learn the reverse of the forward diffusion. A neural network, usually a U-Net [29], is trained to predict the noise that was introduced at a step t: given a noisy sample x_t, the network predicts what is the noise structure that, if subtracted from x_t, would revert the sample to its less noisy version x_{t-1}, or, in some implementations, the final denoised image x_0. After being trained, the model can be used to generate new data by denoising a random noise sample x_T over T steps, to get a clean sample x_0 that matches the distribution of the dataset used during training.

DMs can be adapted to work with conditional generation, where the denoising process is guided by some context, given as input data together with the noisy sample [30]. This conditioning is used to build models that can generate image depending on a prompt or increase the definition of an image given a low-resolution sample. The task of inpainting is one example of using a conditional DM, in which the goal consists of denoising *masked parts* of an image given the non-masked parts. In the context of charge stability diagrams (CSDs), we treat the task of complete missing measurement data as an inpainting problem, where masked parts of the diagram are reconstructed based on the surrounding available data.

6.3.1. MODEL AND MASKING STRATEGY

In this work, we adapt a diffusion-based inpainting model originally developed for reconstructing missing regions in texture, line drawing, and material images [31]. The original model was trained using randomly erased regions that simulate the effect of an eraser removing a stain through back-and-forth motions, resulting in irregular, stochastic mask patterns.

However, our use case diverges significantly. The goal here is to accelerate the acquisition of CSDs by training a diffusion model to reconstruct voltage slices from only partially measured data. To better match this practical setting, we introduce a structured masking scheme designed to resemble realistic experimental measurement protocols.

We propose two types of grid-like masks that simulate an experimental measurement scheme with significantly reduced amount of measured data, while still providing sufficient context for the diffusion model to infer missing values:

- **Reduced Pixel Mask:** In this scheme, only every m-th pixel along each spatial dimension is retained, while the remaining pixels are masked out. This results in a dotted-grid pattern across the image and simulates a coarse scan of the full CSD, significantly reducing the total number of measurements required. The mask is generated using:

```
mask[0, ::reduce_factor, ::reduce_factor] = 0.0
```

- **Line-Cut Mask:** This scheme retains square regions of the CSD by masking out horizontal and vertical stripes of configurable spacing and thickness. This simulates a measurement strategy where only selected “windows” of the image are measured. The mask has a checkerboard-like appearance and is created by:

	Model 1	Model 2	Model 3	Model 4	Model 5
Final Train. Acc. (%)	89.06	76.56	89.06	90.62	93.75
Final Val. Acc. (%)	86.42	78.40	84.57	85.80	78.40

```
mask[:, max(0, top):min(h, bottom), :] = 0.0  # horizontal cuts
mask[:, :, max(0, left):min(w, right)] = 0.0  # vertical cuts
```

These structured masks are specifically designed to reflect plausible experimental procedures, allowing us to train a model that can reconstruct full CSDs from significantly fewer direct measurements. Examples of masked CSDs with the two schemes proposed are shown in Figure.

6.3.2. Experiments

Building on the masking schemes described above, we conduct a comprehensive grid search over key hyperparameters to assess their effects on inpainting performance. We test different batch size, learning rate and number of training steps, along with size of the denoising U-Net by changing the number of channels in the convolutional layers.

We adjust the number of diffusion steps, trading off reconstruction fidelity against inference speed, where fewer steps translate directly into faster CSD completion and hence reduced measurement time.

Simultaneously, connecting to the work on classifying CSDs to build high-quality datasets, we evaluate models trained on dataset obtained different classifier thresholds, which returns CSD datasets of varying size and characteristic features: high-threshold sets contain only the cleanest and conventional diagrams, while lower thresholds introduce greater diversity and noise.

6.4. CSD Classifier Performance and Dataset Characteristics

The five models obtained approximate training accuracies between 75% and 93%, and validation accuracies between 78% and 86%. Individual models' accuracies and showed in Table 1, and training/validation accuracy and loss as a function of training step shown in Figure 2.

The accuracy/loss results show that each network is individually capable of classifying the CSDs with high accuracy (around 80% when analyzing validation values), with a small difference between final accuracies of each model suggesting robustness of the training procedure. By looking at the training curve, one can infer that the models also gradually improve (by having increasing accuracy and decreasing loss throughout training).

These individual results are important for the ensemble learning scheme, where using models together can help reducing variance and producing a more robust prediction, but it is only benefited by the ensemble models being individually accurate.

By setting a threshold in the ensemble approach, we can generate different datasets by filtering the 120 thousand CSDs with different levels of accuracy. As expected, we ob-

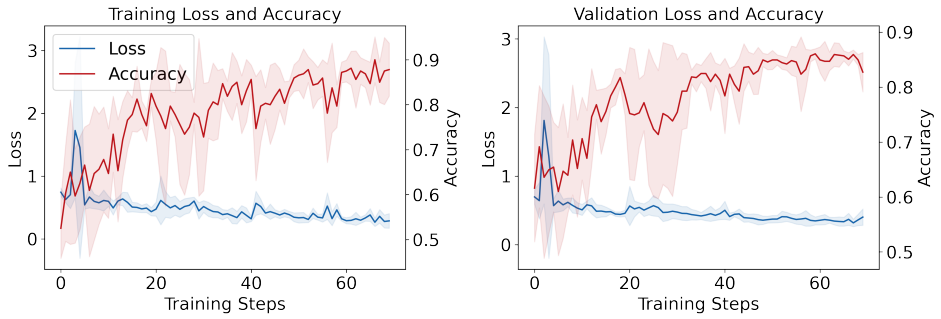


Figure 6.1: Loss (blue) and accuracy (red) for training and validation dataset as a function of training step. Average values are showed as solid lines, with standard deviation presented as shaded area.

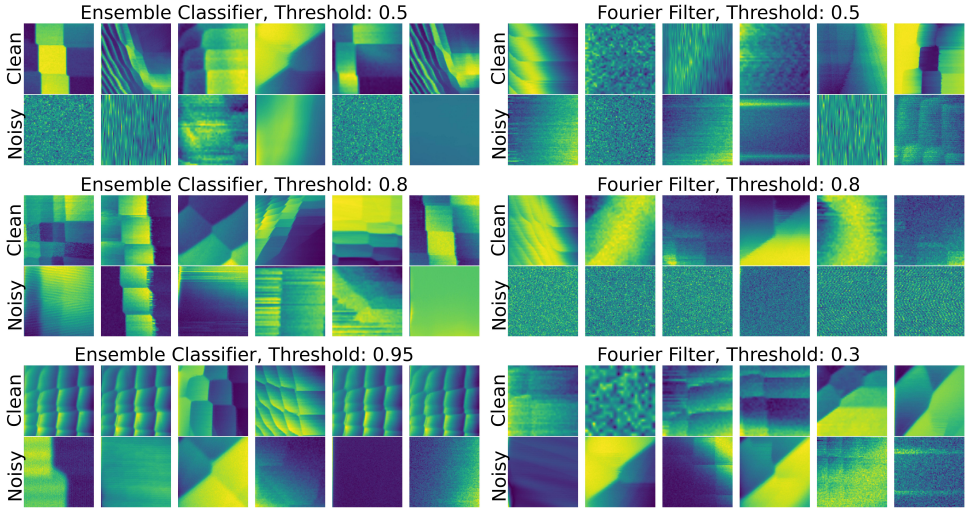


Figure 6.2: Visualization of 6 clean (top row) and noisy (bottom row) samples, for datasets generated using the machine learning ensemble classifier (left) and Fourier filter (right).

served distinct patterns in the composition of the resulting dataset. A threshold of 0.95 resulted in a dataset of 593 images. As shown in Figure 3, one can see that these figures included only extremely clear CSDs, with barely any noise. Moreover, the dataset include CSDs with very well defined features, which can be positive if one is looking for specific patterns, but lacks diversity. Relaxing the threshold to 0.8 significantly expanded the dataset to 9820 images. While still of high quality, this dataset include CSDs with very different features from those present in the 0.95-dataset. Further reducing the threshold to 0.5 resulted in a dataset of 28234 images, and a threshold of 0.3 pushed the size to 34094 images. These larger datasets contained a wider variety of CSDs, including examples with more complex configurations, and some level of noise that human labelers might have categorized as “intermediate”, but still usable for downstream tasks.

These results show that it is possible to systematically build datasets including CSDs with different features by setting the threshold of the ensemble classifier. This makes the approach more robust and appealing when comparing to filter out "noisy" CSD using a Fourier method, for example. This last case is shown at the bottom of Figure 3, where one can see that while the FFT is capable of picking noisy samples, some extremely noisy CSDs are still included in the "clean" dataset.

6.5. Diffusion Model Performance

6.5.1. Training Convergence

In total, we tested 284 hyperparameters combinations, changing learning rate between 0.001 and 0.01; batch size between 4 and 512; diffusion steps between 10 and 1000, training epochs between 10 and 1000; number of channels of the inner convolutional layers of the U-Net between 32 and 96; and training for different datasets, for threshold equal 0.95 with 100 samples, threshold equal to 0.8 and selecting 1000 samples, and threshold 0.5 and 10000 samples. By analyzing all the runs, we found that using Adam as optimizer with learning rate equals to 0.003 returned a smooth training loss, independently of the batch size chosen. Training for 10 steps returned poor results, and 1000 was found not necessary to achieve good performance, so we chose to contrain training to 100 epochs.

Independently of the number of epochs and batch size, we save in total 200 models for each training run, and call each of this saving points a training step. In Fig. 6.3 we show results for inpainting a chosen CSD for a model saved at different training steps, for masking using the reduction method with reduce factor equals to 5, and 100 diffusion steps. We can see that the model gets progressively better at the task at each training step, with the final inpainted CSD getting less noisy throughout training. Here it is already possible to observe that the inpainted image has an artifact at the edges, which is a behavior present in all models trained, and will be disconsidered in the remaining of the chapter, focusing on the "bulk" of the image, and leaving further discussions to the conclusion section. Aside from the artifact, we can see that the result at the end of training is a successful reproduction of the original CSD, especially considering that the main important features for tuning, the phase transitions between different charge states, are present in the synthetic image, and that the mask used occluded 96% of the image.

6.5.2. Diffusion Steps and Dataset Dependence

Since one of the final goals of our application is to speed up the CSD measurement routine, so that more data can be collected more efficiently, the speed of the inpainting process is important, and the number of diffusion steps drastically influences speed, since the more diffusion steps, the more inference steps have to be run. While we first tested using 1000 diffusion steps, training time increases drastically, without significant improvements compared to fewer steps. Given this, we chose to focus on 10 and 100 diffusion steps. In Figures 6.4, 6.5, and 6.6, we show three examples of CSDs inpainted using models trained for different datasets and different diffusion steps, for the reduced mask with reduce factor equals 3 (which occludes 88% of the image), and for the line cut mask with 4 lines and thickness equal to 8 pixels (which occludes X% of the image) in Figures 6.7, 6.8, and 6.9. All the models were trained for 100 epochs with batch size

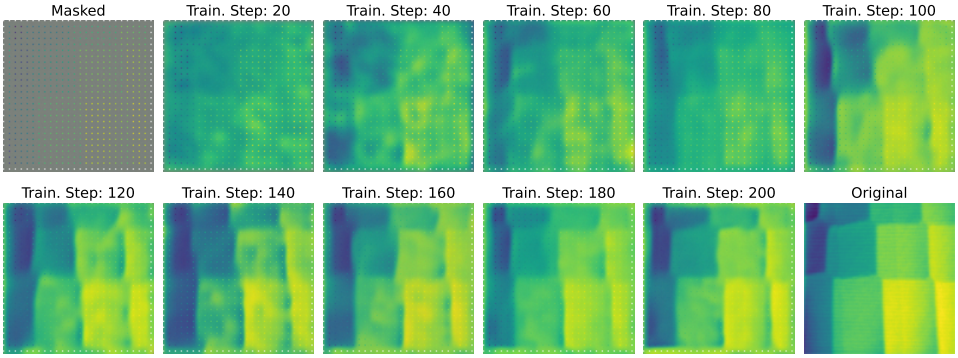


Figure 6.3: Visualization of one inpainted CSD at different stages of training, accompanied by masked CSD and original CSD.

equal to 128, Adam as optimizer and learning rate equals to 0.003, using 32 channels in the inner convolutional layers of the U-Net.

It is visible that for both mask tested, independently of the datasets used to train the models, 10 diffusion steps is not enough to perform high-quality denoising of CSDs (top row in all figures). The only examples that show sufficiently clean CSDs are when training using the dataset obtained with threshold equal to 0.8, especially for reduced mask, like it can be seen in Figures 6.4, 6.5, and 6.6. For the line cut mask, it seems to work for one example, in Fig. 6.7, but not for other cases (Figs. 6.8 and 6.9). This could be improved by implementing better noise schedulers, given that for all the models tested here, we used a simple linear decay in the noise schedule. Regardless, 100 diffusion steps consistently return high-quality CSDs, especially for the reduced-mask case, while not increasing drastically the inference time. For this reason, from now on, we focus the discussion to the cases of 100 diffusion steps.

It is clear from the comparison between the masks, that it is more challenging to obtain sharp and clean CSDs when using the line cut mask: while Fig. 6.7 shows a pristine example, Figures 6.8 and 6.9 contain clear artifacts in the form of noisy squares in certain regions of the image. The reason for this contrasting results is probably because the example inpainted in Fig. 6.7 is part of the high-threshold filtered dataset, which contain the main features selected by the ensemble classifier, while the cases of Figures 6.8 and 6.9 are more complex.

6.5.3. INCREASED DATA OCCLUSION

After seeing the potential of obtaining high-quality images containing phase transitions and main features of the original CSDs, we wanted to test if the results are consistent when increasing the percentage of occluded data, which in the experiment setting would mean measuring fewer data to build a partial CSD and feed it to the diffusion model. In Figures 6.10, 6.11, and 6.12, we show results for inpainting using the reduced mask with reduce factor equals to 5, which means 96% of the original image occluded, and with the line cut mask with 4 lines 4-pixel thick in each direction, occluding X% of the original data. All the models diffuse for 100 steps, and were trained for 100 epochs with batch

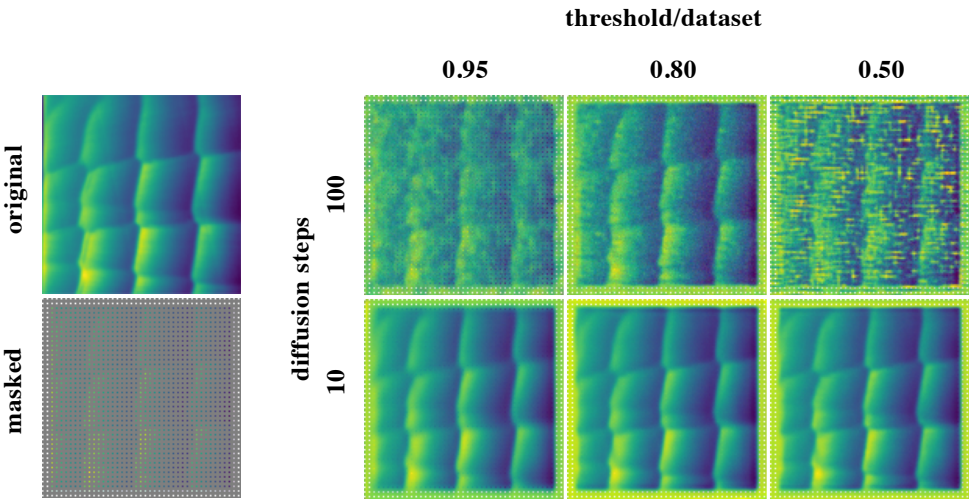


Figure 6.4: Result for inpainted CSDs masked using the reduce method, for models with different diffusion steps, and trained on different datasets, generated with thresholds for the ensemble classifier.

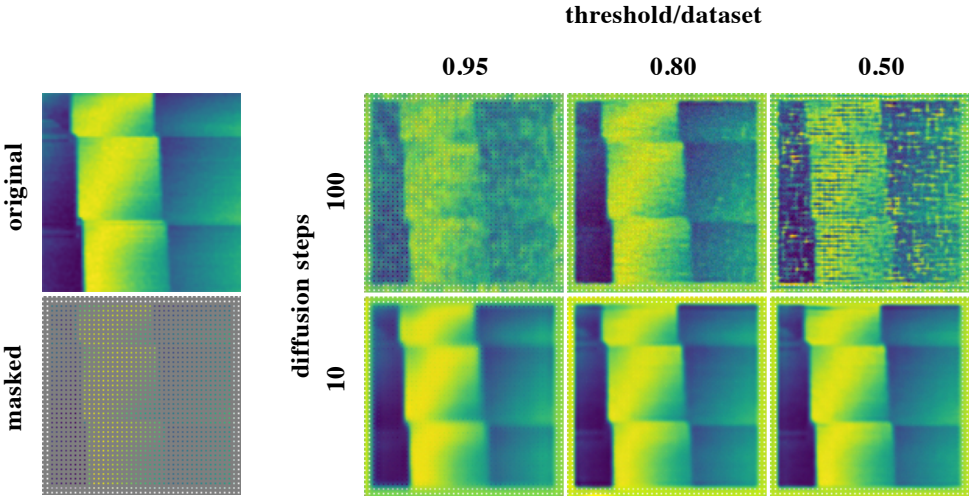


Figure 6.5: Same as Fig. 6.4, for a different CSD sample.

size equal to 128, Adam as optimizer and learning rate equals to 0.003, using 96 channels in the inner convolutional layers of the U-Net.

We can see that, independently of the training set, the model returns high-quality results, with little noise and main features of the original image recovered, when the mask used is the reduced one, while all results for the line cut mask are very poor, with most of the output image being noisy. Alongside the previously showed examples, this indicates that inpainting the line cut mask is a significantly harder task than the reduced case. This makes sense, since the context the model has available to inpaint (the non-

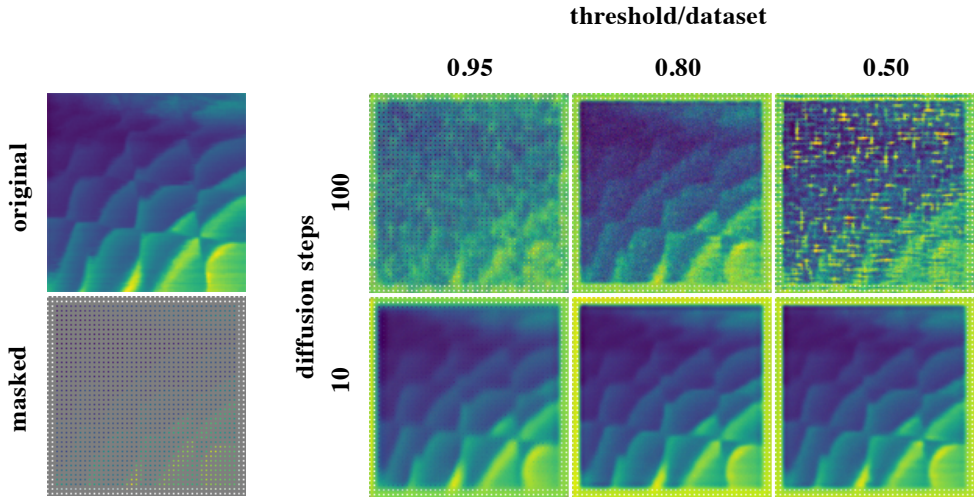


Figure 6.6: Same as Fig. 6.4, for a different CSD sample.

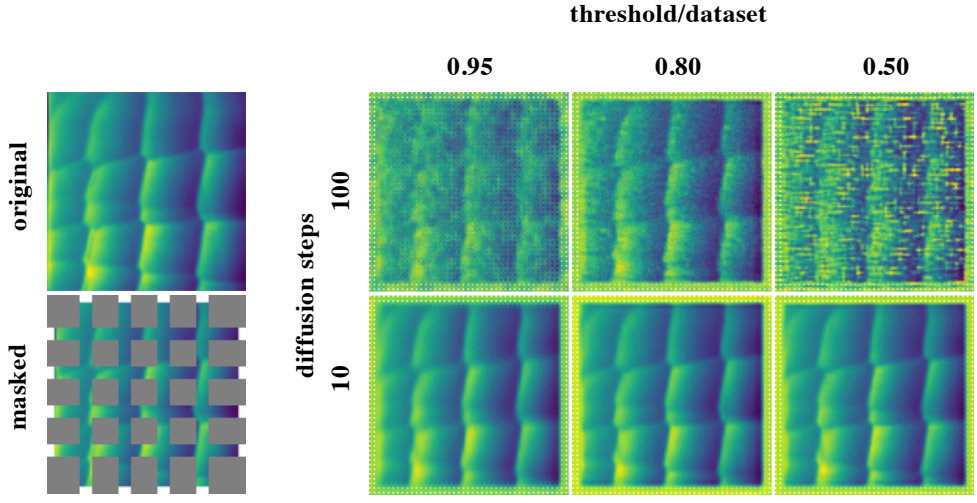


Figure 6.7: Result for inpainted CSDs masked using the line cut method, for models with different diffusion steps, and trained on different datasets, generated with thresholds for the ensemble classifier.

occluded pixels) are sparsely distributed throughout the full image in the reduced mask case, while the line cut mask creates larger regions of occluded data, which creates a large distance between some of the occluded pixels and available context.

6.5.4. TIME PERFORMANCE

As mentioned before, an intended application of our approach is to implement trained models online in experimental measurement routines, speeding up measurement time by only measuring partial CSDs and inpainting them with the diffusion model. For this,

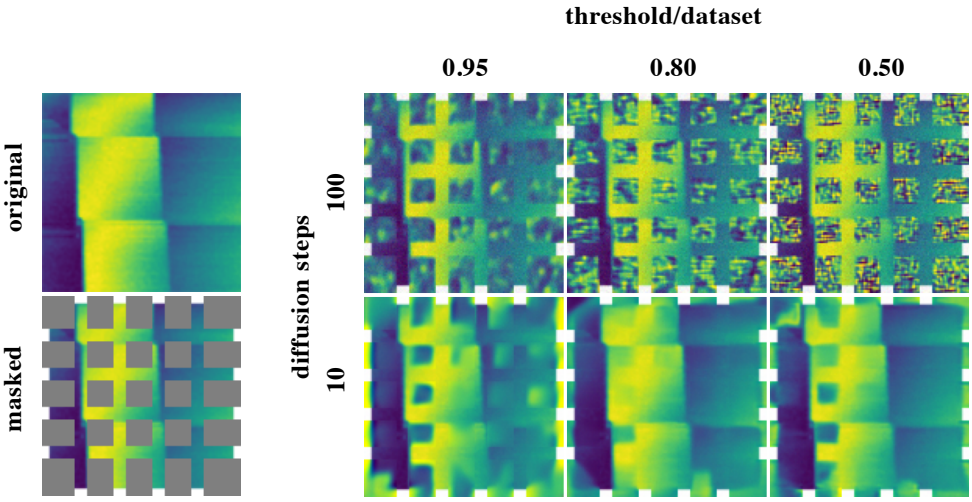


Figure 6.8: Same as Fig. 6.7, for a different CSD sample.

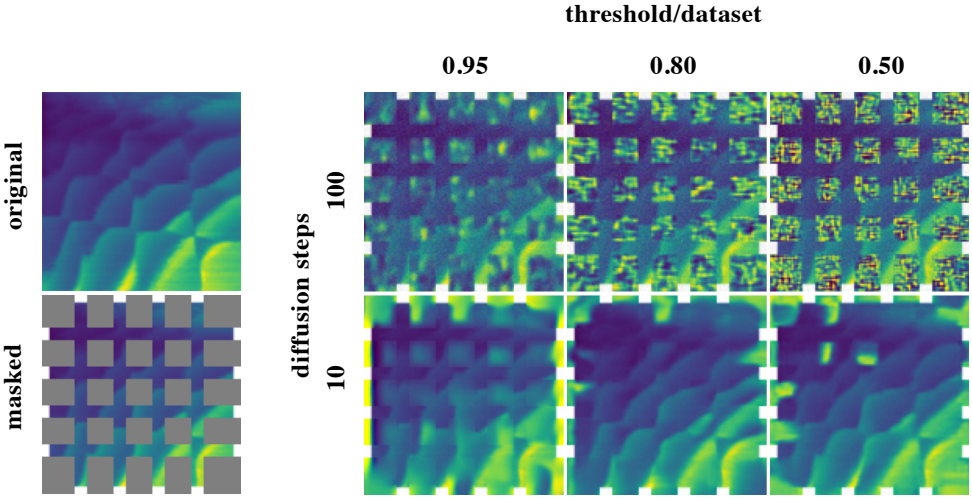


Figure 6.9: Same as Fig. 6.7, for a different CSD sample.

it is crucial that the inference time of inpainting one image is faster than measuring it. Especially, the approach shows itself advantageous if considering the time to measure the partial CSD plus the inpainting time is still (significantly) lower than performing the measurement of the complete CSD.

For this reason, we analyze the inference time of inpainting a CSD with a diffusion model using 100 diffusion steps, throughout different platforms. In Fig. 6.13 we show the average inpainting time using an M1 processor, and three different GPUs: V100S, A100-Small (A100 divided into 8 nodes) and A100, alongside with the time taken to measure a CSD with the same number of points/pixels as the inpainted image (128 x 128), for

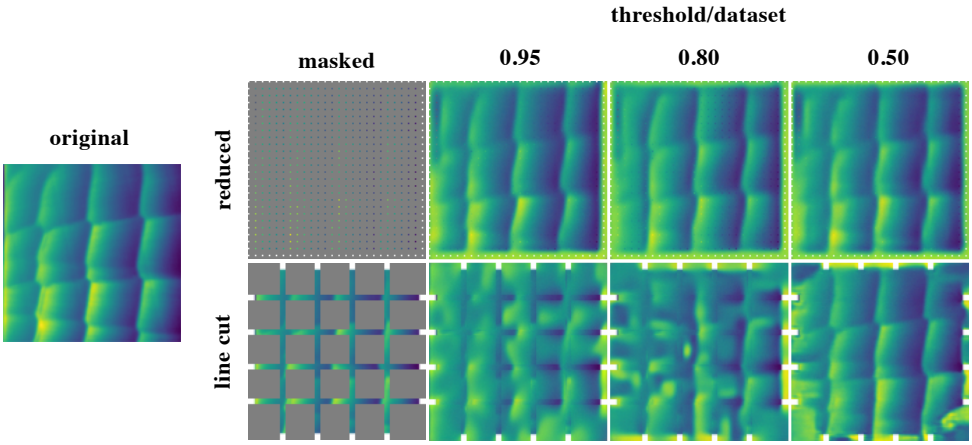


Figure 6.10: Result for inpainted CSDs masked using the reduce method (top row) and line cut method (bottom row), for models with 100 diffusion steps, and trained on different datasets, generated with thresholds for the ensemble classifier.

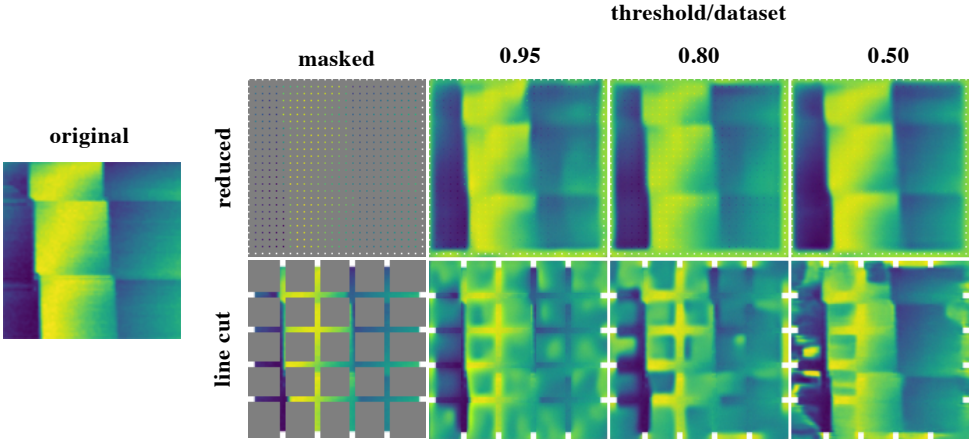


Figure 6.11: Same as Fig. 6.10, for a different CSD sample.

different integration times in RF-spectroscopy. All the GPUs take less than half the time of measuring a full CSD using 20μs integration time, which is close to an usual value used across different labs. This means that, , it is significantly faste to measure the partial data and performing inpainting than to measure the full CSD. The case where inpainting is performed using the reduced mask with reduce factor equals to 5, in which 96% of the data is occluded, for example, means that a full measurement with 20μs integration time would take approximately 0.3s, while measuring only 4% of the pixels and inpainting with a A100 GPU, takes approximately 0.08s, almost a four-fold speedup. This advantage can be even higher by using faster GPUs, and significantly improved by training models that use 10 diffusion steps.

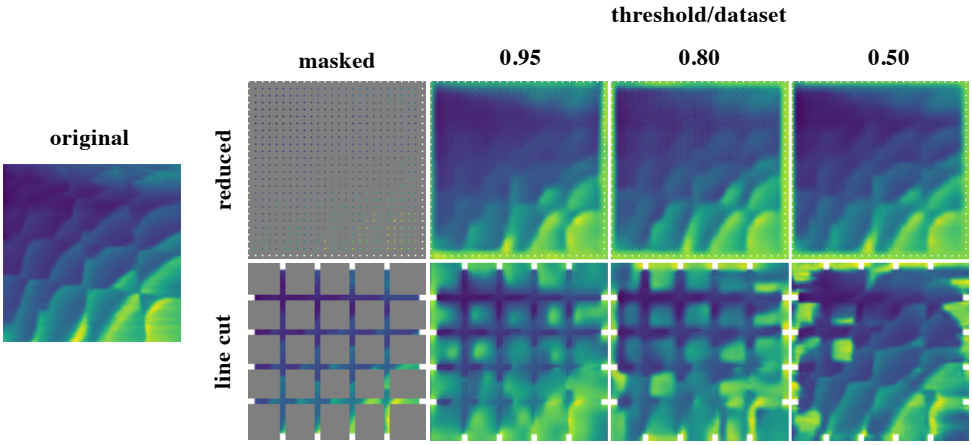


Figure 6.12: Same as Fig. 6.10, for a different CSD sample.

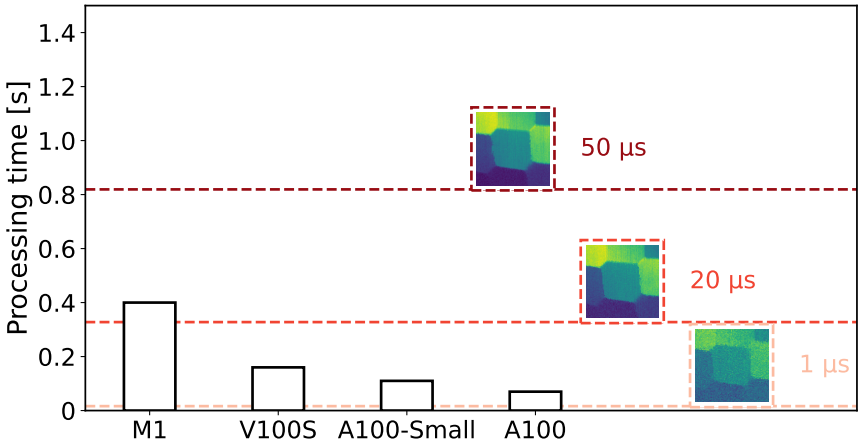


Figure 6.13: Running time of inpainting one CSD using different processors (bars), compared to time to measure the CSD with the same size using RF-spectroscopy with different integration times (colored lines and inset images).

6.6. Conclusion

In this work, we've demonstrated a robust, machine learning-driven framework designed to accelerate and enhance quantum dot experiments, specifically focusing on the critical and often time-consuming task of tuning. Our approach addresses two key challenges: curating high-quality experimental datasets and reducing measurement time through data reconstruction.

We first introduced a supervised machine learning methodology for classifying charge stability diagrams based on their quality. By fine-tuning ResNet18 models on a small, human-labeled subset of CSDs, we successfully built an ensemble classifier capable of accurately distinguishing between clean and noisy CSDs. The ability to precisely control the quality and diversity of the CSDs by adjusting the classifier's threshold is a major step towards developing datasets that can be used to train more robust and generalizable automated tuning models.

Building upon this high-quality dataset, we introduced the use of diffusion models for inpainting partially measured charge stability diagrams. We use a grid-like masking strategy that emulates a measurement protocol in which only selected regions of the original CSD are measured, and are then used as input to the diffusion model to be completed. Our results show that diffusion models can successfully reconstruct full CSDs from significantly occluded data, particularly when using a "reduced pixel mask" that retains sparsely distributed information across the image. While the "line-cut mask" proved more challenging, further improvements in noise schedulers and model architectures could potentially address this. Importantly, our time performance analysis clearly indicates that the combined time for partial data acquisition and inpainting is substantially faster than measuring a full CSD, offering up to a four-fold speedup with current hardware.

REFERENCES

[1] Sander de Snoo et al. *Delft Charge Stability Diagram Dataset*. Zenodo, July 2025. DOI: 10.5281/zenodo.16363472. URL: https://doi.org/10.5281/zenodo.16363472.

[2] Daniel Loss and David P DiVincenzo. "Quantum computation with quantum dots". In: *Physical Review A* 57.1 (1998), p. 120.

[3] Guido Burkard et al. "Semiconductor spin qubits". In: *Reviews of Modern Physics* 95.2 (2023), p. 025003.

[4] LMK Vandersypen et al. "Interfacing spin qubits in quantum dots and donors—hot, dense, and coherent". In: *npj Quantum Information* 3.1 (2017), p. 34.

[5] Anasua Chatterjee et al. "Semiconductor qubits in practice". In: *Nature Reviews Physics* 3.3 (2021), pp. 157–177.

[6] Stephan GJ Philips et al. "Universal control of a six-qubit quantum processor in silicon". In: *Nature* 609.7929 (2022), pp. 919–924.

[7] Francesco Borsoi et al. "Shared control of a 16 semiconductor quantum dot crossbar array". In: *Nature Nanotechnology* 19.1 (2024), pp. 21–27.

[8] Ronald Hanson et al. "Spins in few-electron quantum dots". In: *Reviews of modern physics* 79.4 (2007), pp. 1217–1265.

[9] RJ Schoelkopf et al. "The radio-frequency single-electron transistor (RF-SET): A fast and ultrasensitive electrometer". In: *science* 280.5367 (1998), pp. 1238–1242.

[10] N. Ares et al. "Sensitive Radio-Frequency Measurements of a Quantum Dot by Tuning to Perfect Impedance Matching". In: *Phys. Rev. Appl.* 5 (3 Mar. 2016), p. 034011. DOI: 10.1103/PhysRevApplied.5.034011. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.5.034011.

[11] Sandesh S Kalantre et al. "Machine learning techniques for state recognition and auto-tuning in quantum dots". In: *npj Quantum Information* 5.1 (2019), p. 6.

[12] Dominic T Lennon et al. "Efficiently measuring a quantum device using machine learning". In: *npj Quantum Information* 5.1 (2019), p. 79.

[13] Renato Durrer et al. "Automated tuning of double quantum dots into specific charge states using neural networks". In: *Physical Review Applied* 13.5 (2020), p. 054019.

[14] Justyna P Zwolak et al. "Autotuning of double-dot devices in situ with machine learning". In: *Physical review applied* 13.3 (2020), p. 034075.

[15] Jana Darulová et al. "Autonomous tuning and charge-state detection of gate-defined quantum dots". In: *Physical Review Applied* 13.5 (2020), p. 054005.

[16] Stefanie Czischek et al. "Miniaturizing neural networks for charge state autotuning in quantum dots". In: *Machine Learning: Science and Technology* 3.1 (2021), p. 015001.

[17] V Nguyen et al. "Deep reinforcement learning for efficient measurement of quantum devices". In: *npj Quantum Information* 7.1 (2021), p. 100.

[18] Jozef Bucko et al. “Automated reconstruction of bound states in bilayer graphene quantum dots”. In: *Physical Review Applied* 19.2 (2023), p. 024015.

[19] Jonas Schuff et al. “Fully autonomous tuning of a spin qubit”. In: *arXiv preprint arXiv:2402.03931* (2024).

[20] Rouven Koch et al. “Adversarial Hamiltonian learning of quantum dots in a minimal Kitaev chain”. In: *Physical Review Applied* 20.4 (2023), p. 044081.

[21] Y-Y Liu et al. “Radio-frequency reflectometry in silicon-based quantum dots”. In: *Physical Review Applied* 16.1 (2021), p. 014057.

[22] Cesar F Caiafa et al. *Machine learning methods with noisy, incomplete or small datasets*. 2021.

[23] Qianyu Huang and Tongfang Zhao. “Data collection and labeling techniques for machine learning”. In: *arXiv preprint arXiv:2407.12793* (2024).

[24] Dong-Hyun Lee et al. “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks”. In: *Workshop on challenges in representation learning, ICML*. Vol. 3. 2. Atlanta. 2013, p. 896.

[25] Paola Cascante-Bonilla et al. “Curriculum labeling: Revisiting pseudo-labeling for semi-supervised learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 8. 2021, pp. 6912–6920.

[26] Jascha Sohl-Dickstein et al. “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 2256–2265. URL: https://proceedings.mlr.press/v37/sohl-dickstein15.html.

[27] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[28] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[29] 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: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241.

[30] Zheyuan Zhan et al. “Conditional Image Synthesis with Diffusion Models: A Survey”. In: *arXiv preprint arXiv:2409.19365* (2024).

[31] Nicolas Cherel et al. “Diffusion-based image inpainting with internal learning”. In: *2024 32nd European Signal Processing Conference (EUSIPCO)*. IEEE. 2024, pp. 446–450.

7

ADIABATIC FINE-TUNING OF NEURAL QUANTUM STATES ENABLES DETECTION OF PHASE TRANSITIONS IN WEIGHT SPACE

Neural quantum states (NQS) have emerged as a powerful tool for approximating quantum wavefunctions using deep learning. While these models achieve remarkable accuracy, understanding how they encode physical information remains an open challenge. In this work, we introduce adiabatic fine-tuning, a scheme that trains NQS across a phase diagram, leading to strongly correlated weight representations across different models. This correlation in weight space enables the detection of phase transitions in quantum systems by analyzing the trained network weights alone. We validate our approach on the transverse field Ising model and the J1-J2 Heisenberg model, demonstrating that phase transitions manifest as distinct structures in weight space. Our results establish a connection between physical phase transitions and the geometry of neural network parameters, opening new directions for the interpretability of machine learning models in physics.

The results of this chapter have been published as: **V. Hernandes**, T. Spriggs, S. Khaleefah, E. Greplova, "Adiabatic Fine-Tuning of Neural Quantum States Enables Detection of Phase Transitions in Weight Space", Workshop on Neural Network Weights as a New Data Modality of the International Conference of Learning Representations 2025 [1].

7.1. INTRODUCTION AND RELATED WORK

The simulation of quantum many-body systems represents one of the most challenging problems in computational physics, primarily due to the exponential growth of the Hilbert space with system size [2, 3]. Neural Quantum States (NQS) have emerged as a promising approach, leveraging the expressive power of deep learning to represent quantum wavefunctions [4, 5]. In this approach, neural networks serve as ansätze for the wavefunction, mapping quantum states to their corresponding probability amplitudes. While these models give very good results in ground state energy calculations [6, 7, 8, 9], it remains challenging to understand how they encode physical information.

Recent work by Rende et al. [10] introduced a fine-tuning strategy in which a network pretrained near a phase transition is adapted across a phase diagram by updating only the output layer, reducing computational costs while maintaining accuracy across phases. Building on this idea, we investigate how phase transitions relate to the geometry of neural network weight space. In deep learning, various studies have explored the structure of weight space with diverse objectives, ranging from analyzing the loss landscape to understand mode connectivity [11, 12], to leveraging neural network weights as data for downstream predictions [13, 14, 15]. Meanwhile, in quantum physics, phase transitions are traditionally characterized through order parameters and correlation functions. The intersection of these fields has led to emerging research on how neural networks capture physical symmetries and order parameters [16, 17, 18], and how it's possible to exploit machine learning models to learn about physical phase transitions [19, 20, 21, 22]. However, directly analyzing the weights of neural quantum states as a means of detecting phase transitions remains largely unexplored.

In this work, we introduce a fine-tuning scheme, called **adiabatic fine-tuning**, that enables systematic training of NQS across a phase diagram, leading to more accurate models, and highly correlated weights across different models. We show that this correlation can be leveraged to detect phase transitions by analyzing the PCA projection of trained network weights alone. Applying this method to the transverse field Ising model and the J1-J2 Heisenberg model, we demonstrate that phase transitions manifest as distinct geometric patterns in weight space. By directly analyzing the trained network weights, we show how the evolution of neural network parameters reflects physical features of quantum systems, providing a new method to understand phase transitions.

7.2. METHODOLOGY

7.2.1. NEURAL QUANTUM STATE TRAINING

We trained neural networks models to approximate the quantum wavefunctions of the systems under study. Specifically, the wavefunctions for the transverse field Ising model (TFIM) and J1-J2 Heisenberg model were parameterized using neural networks, with the goal of minimizing the energy of the system. We implemented a restricted Boltzmann machine architecture with a single hidden layer, using system sizes of $N = 8$ spins for the Ising model and $N = 12$ spins for the J1-J2 model (see Appendix C for architectural details). In this study, we focus on small system sizes for which we can compute the exact wavefunction, allowing direct comparison to known results.

7.2.2. TRANSVERSE FIELD ISING MODEL

The TFIM in one dimension is defined by the Hamiltonian:

$$\hat{H} = -J\sum_i \sigma_i^z \sigma_{i+1}^z - h\sum_i \sigma_i^x, \tag{7.1}$$

where J is the interaction strength between neighboring spins, and h is the external transverse magnetic field. This model exhibits a quantum phase transition from a ferromagnetic to a paramagnetic phase at the critical field value $|h_c/J| = 1$

7.2.3. J1-J2 HEISENBERG MODEL

The 1D J1-J2 model is described by the Hamiltonian:

$$\hat{H} = J_1\sum_i \mathbf{S}_i \cdot \mathbf{S}_{i+1} + J_2\sum_i \mathbf{S}_i \cdot \mathbf{S}_{i+2}, \tag{7.2}$$

where $\mathbf{S}_i$ represents the spin operator at site i, and J_1 and J_2 define the nearest and next-nearest neighbor interactions, respectively. This model exhibits several distinct magnetic phases depending on the ratio J_2/J_1.

7.2.4. MODEL PERFORMANCE METRICS

The performance of our model in obtaining the correct ground state is assessed by calculating the *energy error* and *infidelity*. The energy error, defined as $|E_{\text{NQS}} - E_{\text{exact}}|$, quantifies the deviation of the neural network-estimated energy from the exact ground state energy. The infidelity, given by $1 - |\langle\psi_{\text{exact}}|\psi_{\text{NQS}}\rangle|^2$, measures the discrepancy between the trained and exact wavefunctions, with lower values indicating better overlap.

7.2.5. FINE-TUNED TRAINING STRATEGY

To explore the structure of weight spaces across phase transitions, we employ two distinct training strategies. The first approach is **independent training**. In this method, each model is initialized randomly and trained separately for each value of the control parameter, h (for the Ising model) or J_2/J_1 (for the J1-J2 Heisenberg model). This strategy allows us to study how the model weights evolve when trained independently for each specific parameter setting. It serves as a baseline for understanding how weight space behaves when there is no continuity between parameter values.

The second approach we call **adiabatic fine-tuning**. In this case, we initialize each new model using the weights of a previously trained model from the neighboring point in the phase diagram. This ensures continuity in the evolution of model parameters, creating a smooth trajectory through weight space that reflects the gradual changes in the underlying quantum system. Fine-tuning allows us to capture the relationship between the geometry of weight space and physical phase transitions in a more connected and coherent manner. By comparing the results from fine-tuned training to those from independent training, we can identify phase transitions as distinct features in the neural network parameter evolution, revealing the structural nature of weight spaces across these transitions.

This fine-tuned training strategy creates a connected trajectory in weight space, which resembles the concept of mode connectivity observed in deep learning. The continuity

in parameter evolution provided by fine-tuning enables us to detect phase transitions in quantum systems as sharp, discernible features in the progression of the network parameters.

7.2.6. Principal Component Analysis of Weights

After training the models, we analyze the resulting weight vectors by performing principal component analysis (PCA). This dimensionality reduction technique identifies the dominant directions in weight space and helps reveal how the weights evolve as the quantum system's control parameters change.

7.3. Results

7.3.1. Transverse Field Ising Model

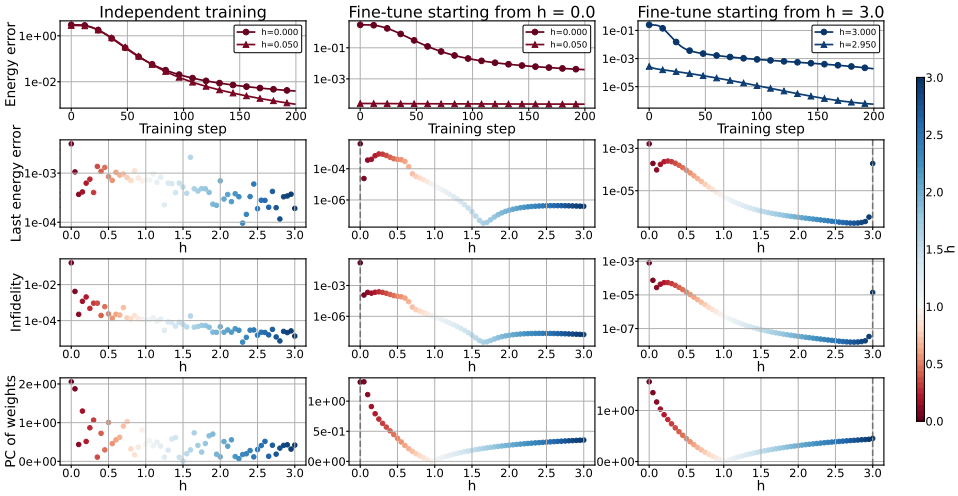


Figure 7.1: Results for the Ising model. From top to bottom the rows show: the energy error over the first two training iterations, the last energy error values for each value of h, the infidelity values, and the principal component analysis of the weights. From left to right columns show the case of independent training, and fine-tuning models starting from $h = 0.0$, and $h = 3.0$. The colorbar represents the value of h.

Our analysis of the transverse field Ising model reveals a clear correspondence between the known quantum phase transition at $h_c = 1$ when $J = -1$ and features in the weight space trajectory. The first principal component shows a pronounced minimum at the critical point, providing a direct signature of the transition from the ferromagnetic to paramagnetic phase, as shown in Figure 7.1.

The energy plots in the top row of Figure 7.1 reveal a key difference between independent training and fine-tuning strategies. In the case of independent training, each model starts at an arbitrarily high energy and gradually converges to a lower value. In contrast, fine-tuning ensures that the energy at the beginning of a new training run is already close to the final energy of the previous one. This continuity in the optimization process sug-

gests that fine-tuned training preserves learned features from previous models, leading to a smoother transition through the phase diagram. Additionally, examining the second and third rows of the figure, we see that the final energy and infidelity values remain close for consecutive fine-tuned runs, except for lower values of the field h, where deviations become more pronounced.

7.3.2. J1-J2 HEISENBERG MODEL

In the J1-J2 model, our method successfully identifies the change in the system's structure at $J_2/J_1 = 0.5$, appearing again as a minimum in the first PCA component, as shown in the bottom row of Fig. 7.2.

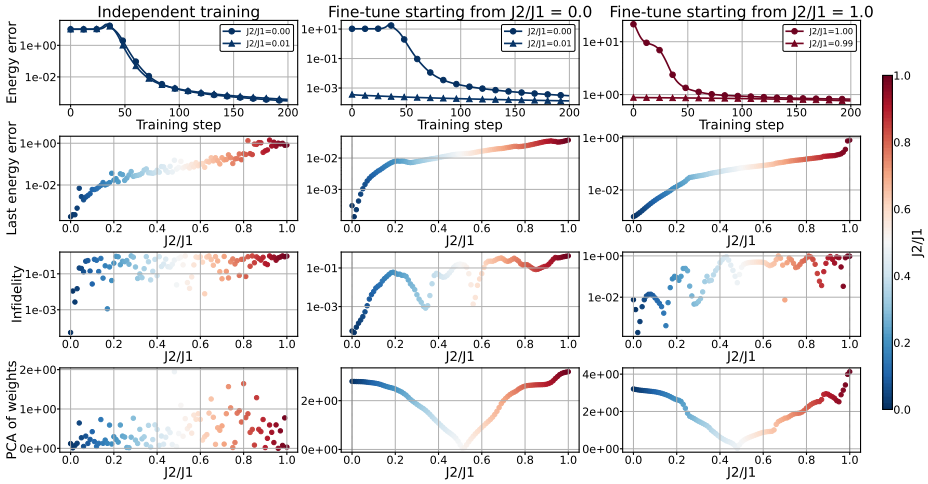


Figure 7.2: Results for the J1-J2 model. From top to bottom the rows show: the energy error over the first two training iterations, the last energy error values for each value of $J2/J1$, the infidelity values, and the principal component analysis of the weights. From left to right columns show the case of independent training, and fine-tuning models starting from $J_2/J_1 = 0.0$, and $J_2/J_1 = 1.0$. The colorbar represents the J_2/J_1 ratio.

The J1-J2 model exhibits the same trends in energy evolution, final energy values, and infidelity as observed in the Ising model, as shown in the top three rows of Figure 7.2.

7.4. DISCUSSION AND CONCLUSIONS

Our work establishes a novel connection between phase transitions in quantum systems and the geometry of neural network weight spaces. The success of our method in detecting known phase transitions suggests that neural networks encode physically meaningful information in their weights in a structured and analyzable way.

This finding has several important implications. First, it provides a new tool for studying quantum phase transitions that doesn't require explicit construction of order parameters or prior knowledge of the relevant physical observables. Second, it offers insights into how neural networks encode physical information, this can be seen from the

change in the behavior of the network's weights across the phase transition.

Future directions include extending this analysis to more complex quantum systems and investigating whether adiabatic fine-tuning can be applied to other domains where neural networks model systems with phase transitions or structural changes. The connection to mode connectivity in deep learning also suggests potential applications in understanding the loss landscapes of neural networks more generally.

7.5. ENERGY FOR ALL MODELS

Figures 7.3 and 7.4 show the energy error as a function of training steps for all values of the control parameters in both models. These plots highlight how the adiabatic fine-tuning strategy maintains lower initial energy errors compared to independent training, demonstrating the advantage of leveraging previously trained weights for improved convergence.

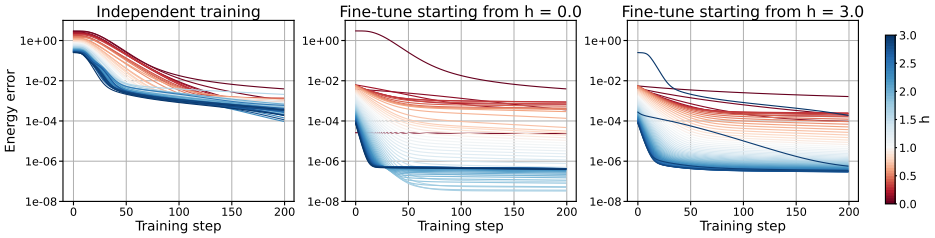


Figure 7.3: Energy error as a function of training step for all values of h in the TFIM model.

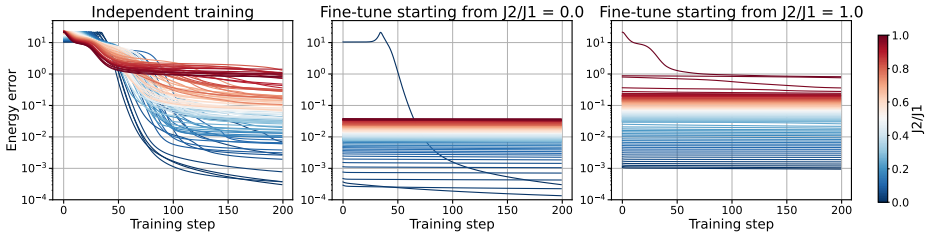


Figure 7.4: Energy error as a function of training step for all values of $J2/J1$ in the J1-J2 model.

7.6. PCA VISUALIZATIONS

Further visualizations of the principal component analysis projections for both models are shown in Figure 7.5. The scatter plots and 3D visualizations provide a clearer understanding of how weights for models trained with different values of h (Ising model) and J_2/J_1 (J1-J2 model) show strong correlation, forming a helix structure when projected in PC space. The existence of a similar pattern for different systems suggests that the fine tuning training scheme results in universal properties to be investigated in future work.

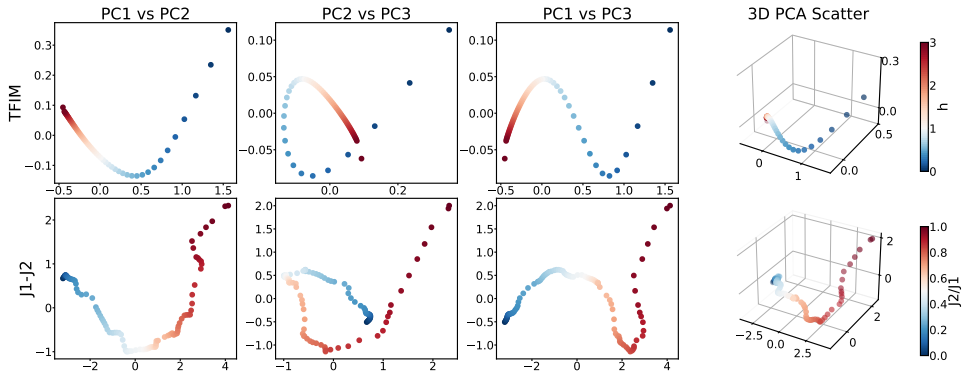


Figure 7.5: Scatter plots and 3D visualizations of PCA projections of the weights of neural networks trained for different values of h in the case of the Ising model (first row), and for different values of J_2/J_1 in the case of the J1-J2 model (second row). From left to right, columns show PC1 vs PC2 space, PC2 vs PC3 space, PC1 vs PC3 spaces, and the 3D PCA projection. The colorbars to the right indicate the corresponding parameter values for each model. The phase transition point for all images is represented by the divergence of the colormap, shown in white.

7.7. NETWORK ARCHITECTURE AND TRAINING DETAILS

Table 7.1: Training hyperparameters

Parameter	Ising Model	J1-J2 Model
Learning rate	0.01	0.01
Training steps	200	200
System size (spins)	8	12
Hidden layer ratio (α)	1	2
Coupling increment (Δh or $\Delta J_2/J_1$)	0.025	0.01
Coupling range	$[0,3]$	$[0,1]$
Network weights type	Real	Complex

Our neural network architecture consists of a restricted Boltzmann machine (RBM) with a single hidden layer using logcosh activation function, implemented as follows:

$$\psi(x) = \sum_i \text{logcosh}(W_i x + b_i) \tag{7.3}$$

where W_i represents the weights connecting the input to the hidden layer, and b_i are the bias terms. The number of neurons in the hidden layer is determined by αN, where N is the number of input neurons (spins) and α is the hidden layer ratio. For the Ising model, we used real-valued weights, while for the J1-J2 model, we employed complex-valued weights (with PCA performed on the real components only). Hyperparameters used to trained models for both systems are shown in Table 1.

REFERENCES

[1] Vinicius Hernandes et al. “Adiabatic Fine-Tuning of Neural Quantum States Enables Detection of Phase Transitions in Weight Space”. In: *Workshop on Neural Network Weights as a New Data Modality*.

[2] Richard P Feynman. “Simulating physics with computers”. In: *Feynman and computation*. cRc Press, 2018, pp. 133–153.

[3] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[4] Giuseppe Carleo and Matthias Troyer. “Solving the quantum many-body problem with artificial neural networks”. In: *Science* 355.6325 (2017), pp. 602–606.

[5] Matija Medvidović and Javier Robledo Moreno. “Neural-network quantum states for many-body physics”. In: *The European Physical Journal Plus* 139.7 (2024), pp. 1–26.

[6] Ao Chen and Markus Heyl. “Efficient optimization of deep neural quantum states toward machine precision”. In: *arXiv preprint arXiv:2302.01941* (2023).

[7] Agnes Valenti et al. “Correlation-enhanced neural networks as interpretable variational quantum states”. In: *Physical Review Research* 4.1 (2022), p. L012010.

[8] Dian Wu et al. “Variational benchmarks for quantum many-body problems”. In: *Science* 386.6719 (2024), pp. 296–301.

[9] Riccardo Rende et al. “A simple linear algebra identity to optimize large-scale neural network quantum states”. In: *Communications Physics* 7.1 (2024), p. 260.

[10] Riccardo Rende et al. “Fine-tuning neural network quantum states”. In: *Physical Review Research* 6.4 (2024), p. 043280.

[11] Timur Garipov et al. “Loss surfaces, mode connectivity, and fast ensembling of dnns”. In: *Advances in neural information processing systems* 31 (2018).

[12] Rahim Entezari et al. “The role of permutation invariance in linear mode connectivity of neural networks”. In: *arXiv preprint arXiv:2110.06296* (2021).

[13] Gabriel Eilertsen et al. “Classifying the classifier: dissecting the weight space of neural networks”. In: *ECAI 2020*. IOS Press, 2020, pp. 1119–1126.

[14] Thomas Unterthiner et al. “Predicting neural network accuracy from weights”. In: *arXiv preprint arXiv:2002.11448* (2020).

[15] Konstantin Schürholt, Michael W Mahoney, and Damian Borth. “Towards scalable and versatile weight space learning”. In: *arXiv preprint arXiv:2406.09997* (2024).

[16] Sebastian J Wetzel et al. “Discovering symmetry invariants and conserved quantities by interpreting siamese neural networks”. In: *Physical Review Research* 2.3 (2020), p. 033499.

[17] Felix Frohnert and Evert van Nieuwenburg. “Explainable representation learning of small quantum states”. In: *Machine Learning: Science and Technology* 5.1 (2024), p. 015001.

[18] Kacper Cybiński et al. “Speak so a physicist can understand you! TetrisCNN for detecting phase transitions and order parameters”. In: *arXiv preprint arXiv:2411.02237* (2024).

[19] Evert PL Van Nieuwenburg, Ye-Hua Liu, and Sebastian D Huber. “Learning phase transitions by confusion”. In: *Nature Physics* 13.5 (2017), pp. 435–439.

[20] Anna Dawid et al. “Phase detection with neural networks: interpreting the black box”. In: *New Journal of Physics* 22.11 (2020), p. 115001.

[21] Eliska Greplova et al. “Unsupervised identification of topological phase transitions using predictive models”. In: *New Journal of Physics* 22.4 (2020), p. 045003.

[22] Julian Arnold et al. “Interpretable and unsupervised phase classification”. In: *Physical Review Research* 3.3 (2021), p. 033052.

8

OUTLOOK

This chapter explores some possible future directions emerging at the intersection of artificial intelligence and nanoscience, building upon the advancements presented in the thesis.

8.1. NQS-ASSISTED QUANTUM STATE RECONSTRUCTION

Neural quantum states (NQS) provide a powerful ansatz to approximate many-body wave functions. We saw in Chapter 7 that it is possible to fine-tune neural networks across a phase diagram of a quantum system, and detect where the phase transition happens purely by analyzing the weights of the trained networks. A natural direction of this work involves testing the technique for more complex systems than the one-dimensional transverse field Ising model (TFIM) , such as two-dimensional systems, fermionic systems, and systems that exhibit topological phase transitions. Furthermore, while principal component analysis provided effective for detecting the phase transition in the example of the TFIM, more sophisticated techniques from the weight-space learning literature [1, 2] could be implemented to obtain a more robust and systematic phase transition detection algorithm, like training a different neural network to classify the system's phase based on the weights of the NQS models.

A different framework in which NQS are increasingly being utilized is quantum state tomography (QST), which consists in the reconstruction of quantum states from experimental measurements [3]. In quantum computing, accurately characterizing a quantum state as the number of qubits grows quickly becomes an untractable task, due to the exponential scaling of the Hilbert space with the system size. To circumvent this, strategies such as classical shadow tomography, which employs random measurement protocols and classical post-processing to efficiently estimate observables, have been developed [4]. Alternatively, one can use a specific ansatz to approximate a quantum state based limited experimental data.The fitted ansatz can then be used to generate new samples or predict properties of the full quantum state by efficiently estimating observables. Early explorations in this domain have utilized tensor networks, and, more recently, NQS [5, 6].

The first work to propose NQS for QST implemented an RBM to fit ground-state wavefunction and dynamical states for simulations of up to 100 qubits [5]. Following works cleverly adapted this protocol so that a product of single-qubit Positive Operator-Valued Measures (POVMs) is used to train the RBM, and also tested an RNN as the NQS neural network architecture [7]. This combination of methods was coined POVM-NQS and it was applied to reconstruct states like GHZ, and states of spin models in one and two dimensional lattices. One clear advantage of reconstructing states using NQS lies in their ability to efficiently calculate observables of the system that would otherwise necessitate an prohibitively large number of measurements. This capability was implemented by Torlai and collaborators for quantum chemistry Hamiltonians [8]. Later, Iouchtchenko and collaborators have further investigated the aspect of sample complexity, assessing the number of independent measurements required to reproduce system properties like energy or fidelity within a given accuracy [9]. More recently, the sample complexity for the specific case of POVM-NQS for mixed-state reconstruction has been explored in depth [10]. Simultaneously, the field has seen the implementation of transformer-based NQS architectures for reconstructing the states of experimental Rydberg atom systems [11], demonstrating the versatility and power of these advanced neural network designs in real-world quantum experiments.

A possible future direction, bridging the approach introduced in Chapter 7 with this experimental NQS setting, is implementing weight-space learning techniques to NQS

models trained on experimental data. This could be used to try to extract features of the reconstructed quantum state form the weights of the ansatz alone, of even to assess properties of the system without the need of calculating observables.

Another significant path for future research involves a deeper understanding of the energy landscape of NQS models. A more comprehensive knowledge of this landscape could guide the design of more tailored and efficient NQS architectures. Recently, groups interested in restricted Boltzmann machines (RBMs) as NQS ansatzes have started applying physically-informed tools to investigate their landscape properties [12]. Our plan is to employ techniques from the broader machine learning literature on visualizing loss landscape and training dynamics [13, 14, 15], to gain deeper insight into how NQS learn to represent complex quantum states. In Chapter 7 we demonstrated that there is a clear mark of the phase transition in the principal components of the weights of trained NQS; understanding how the weights evolve for different physical regimes, depending features of the energy landscape, such as curvatures or the presence of valleys, could provide a powerful tool to study physical systems of interest. This knowledge can be exploited to design optimal training strategies that depend on features of the physical system, and as a probe to gain deeper knowledge about quantum models.

8.2. QUANTUM-ASSISTED COMPUTATIONAL NEUROSCIENCE

In Chapter 2 we introduced the use of quantum generative adversarial networks (QGAN) to generate synthetic neuronal activity data, and believe this intersection between quantum computing and computational neuroscience is a fertile ground for scientific advancements. While the statistical fit showed by this approach is not on par with classical counterparts, we argued that the latter needs to development of tailored architectures that exploit symmetries of neuronal data, while our quantum model achieved significant results using a general architecture with a reduced number of parameters. One straightforward expansion of this work consists in implementing more efficient and easily trainable quantum machine learning models, and verify if including symmetries improve the statistical fit to a level close to that obtained when using classical machine learning models. Developing a model that achieves this while maintaining a low number of trainable parameters would significantly improve the practical utility of quantum techniques in neuroscience.

Another direction consists in using quantum models to generate synthetic that is considerably more complex than the salamander retina dataset, which is often considered a simplistic benchmark. This include datasets similar to that used in Chapter 4: an experimental dataset with intricate spatio-temporal features, like network bursts. Here, an opportunity to integrate Chapter 3 and Chapter 4 is also present: one could use the autoMEA tool to predict burst activity within synthetic data generated by a quantum model. This cross-validation, comparing predictions on synthetic quantum-generated data with results from experimental data, would serve as a powerful method to validate the accuracy of quantum generative models in capturing complex neuronal phenomena.

8.3. AI-ASSISTED CSD MEASUREMENT

The characterization and tuning of quantum dot devices is a crucial step in experimental quantum computing, but comes with several challenges, from the scalability of controls with system size, to the necessity of a human-operator in the tuning loop. In Chapter 2 we introduced QDSim, a computationally efficient simulator of synthetic charge stability diagrams (CSD), based on device specification and geometry. In Chapter 6, we presented a machine learning based technique to build high-quality CSD datasets, and a diffusion model application to reconstruct full CSDs based on partial measurements. Integrating ideas from both chapter open several scientific directions.

One possible path is to run several simulations using QDSim, for systems with different device features, changing geometry (quantum dot and gate positions) and physical properties (dots and gate capacitances), and classify the outcome using the ensemble-classifier. This technique could be interpreted as a filter of device features that return CSDs indicative of well-formed quantum dots, assisting in device design.

Another promising approach is to train diffusion models trained on QDSim generated data. We showed that the ensemble classifier, depending on the threshold used, can filter CSDs with different features, such as specific charge transitions. This allows a strategy where QDSim is used to generate CSD with very specific features, which can then be utilized to enhance the training of the diffusion models. For example, if experimental data for CSDs with certain features are rare or scarce, QDSim could be used to create a synthetic dataset that exhibit these features. This targeted data augmentation would allow diffusion models to learn a more robust representation of the CSD manifold, improving their ability to accurately inpaint or complete CSDs even under challenging experimental conditions.

REFERENCES

[1] Gabriel Eilertsen et al. “Classifying the classifier: dissecting the weight space of neural networks”. In: *ECAI 2020*. IOS Press, 2020, pp. 1119–1126.

[2] Konstantin Schürholt, Michael W Mahoney, and Damian Borth. “Towards scalable and versatile weight space learning”. In: *arXiv preprint arXiv:2406.09997* (2024).

[3] Matthias Christandl and Renato Renner. “Reliable quantum state tomography”. In: *Physical Review Letters* 109.12 (2012), p. 120403.

[4] Scott Aaronson. “Shadow tomography of quantum states”. In: *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*. 2018, pp. 325–338.

[5] Giacomo Torlai et al. “Neural-network quantum state tomography”. In: *Nature physics* 14.5 (2018), pp. 447–450.

[6] Hannah Lange et al. “From architectures to applications: A review of neural quantum states”. In: *Quantum Science and Technology* (2024).

[7] Juan Carrasquilla et al. “Reconstructing quantum states with generative models”. In: *Nature Machine Intelligence* 1.3 (2019), pp. 155–161.

[8] Giacomo Torlai et al. “Precise measurement of quantum observables with neural-network estimators”. In: *Physical Review Research* 2.2 (2020), p. 022060.

[9] Dmitri Iouchtchenko et al. “Neural network enhanced measurement efficiency for molecular groundstates”. In: *Machine Learning: Science and Technology* 4.1 (2023), p. 015016.

[10] Haimeng Zhao, Giuseppe Carleo, and Filippo Vicentini. “Empirical sample complexity of neural network mixed state reconstruction”. In: *Quantum* 8 (2024), p. 1358.

[11] David Fitzek et al. “RydbergGPT”. In: *arXiv preprint arXiv:2405.21052* (2024).

[12] J Quetzalcóatl Toledo-Marin et al. “Exploring the Energy Landscape of RBMs: Reciprocal Space Insights into Bosons, Hierarchical Learning and Symmetry Breaking”. In: *arXiv preprint arXiv:2503.21536* (2025).

[13] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. “Qualitatively characterizing neural network optimization problems”. In: *arXiv preprint arXiv:1412.6544* (2014).

[14] Eliana Lorch. “Visualizing deep network training trajectories with pca”. In: *ICML Workshop on Visualization for Deep Learning*. 2016.

[15] Hao Li et al. “Visualizing the loss landscape of neural nets”. In: *Advances in neural information processing systems* 31 (2018).

9

CONCLUSION

This thesis constitutes an interdisciplinary exploration of the power of artificial intelligence when applied to diverse domains within nanoscience. This work not only significantly contributes to the development of methodologies that increase our understanding of physical and biological systems, but also streamline the processes of experimental analysis and device tuning. The overall topic has been to build strong connections between theoretical models and empirical observations, using artificial intelligence as a unifying tool.

In the domain of computational neuroscience, we obtained significant advancements by implementing a quantum machine learning approach to model neuronal activity, and an AI-assisted experimental data analysis routine. In Chapter 3, we showed how quantum generative adversarial networks can be used to generate synthetic neuronal activity data. While early-stage, this work highlighted the potential of quantum approaches to capture intricate statistical correlations in neural data with potentially fewer parameters. Chapter 4 presented autoMEA, an AI-driven solution for the automated and robust detection of neuronal bursts in complex multi-electrode array recordings. This tool significantly reduces manual labor time, providing a pipeline for analyzing high-dimensional data.

In the context of quantum device characterization and tuning automation, this thesis delivered state-of-the-art tools and methodologies for advancing the capabilities of quantum computing hardware. Chapter 5 introduced QDSim, a computationally efficient simulator designed for generating synthetic charge stability diagrams (CSDs) of quantum dot devices. This simulator offers a critical resource for exploring vast design spaces and generating training data for machine learning models. Chapter 6 demonstrated the power of advanced deep learning techniques, including ensemble classifiers for CSD filtering, and diffusion models for inpainting missing data within CSDs. The developed AI models offer a path towards more autonomous and efficient quantum device tuning, mitigating the time-consuming and expertise-intensive manual procedures that currently limit scalability.

On the subject of condensed matter physics, this thesis demonstrated a significant

contribution to the field of neural quantum states (NQS). We showed how to expand the utility of NQS, from an efficient ansatz to represent quantum wavefunctions, to a tool to study the underlying physics of the simulated systems. In Chapter 7, we showed an approach where NQS, when trained in an adiabatically fine-tuning scheme, enabled the detection of quantum phase transitions by analyzing the evolution of their weight-space trajectories. This methodology provides a computationally efficient alternative to traditional order parameters, offering a new perspective on phase detection.

In summary, this thesis demonstrates that artificial intelligence is not only a supplementary tool but a fundamental paradigm shift for nanoscience. By providing new computational techniques for studying complex quantum systems, developing analysis and simulations frameworks for large-scale neuroscience data, and introducing efficient methods for quantum device characterization and tuning, this work showcases connections between theoretical and experimental frameworks in diverse nanoscience domains.

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor, **Eliška**. You have truly built an inclusive research group where everyone feels welcome and heard. Your ability to propose and manage such diverse research topics demonstrates a broad knowledge and deep scientific curiosity, which inspire young researchers to strive for a more equal and fair academic environment.

I would also like to thank my co-promotor, **Ronald**, and the committee members **Juan**, **Anna**, **Menno**, **Leo**, and **Herre** for agreeing to evaluate my thesis and for engaging in a fruitful scientific discussion.

Then I would like to thank my paranymphs, **Rouven** and **Tom**. Thank you for accepting to be there and helping me with the defense. I'm even more grateful for the projects we are developing together, and for the constant laughs during lunch and at bar tables after work.

I would also like to thank all my collaborators and colleagues. Special thanks to **Arash** and **Jin**, who welcomed me into the group when I arrived in the Netherlands in the middle of the pandemic. You made my adaptation to a new university and country much easier, and passed along the values and camaraderie of QMAI.

I'm deeply grateful to all the students I supervised: **Maia**, **Charles**, **Valentina**, **Aram**, **Antón**, **Yicong**, and **Bianca**. I learned much more by supervising you than you learned from me. Thank you for that opportunity.

I could not forget to mention the other members who were and are part of QMAI: **Jan**, **Tanko**, **Naoual**, **Achmed**, **Ignacio**, **Genya**, **Mohammed**, **Cagan**, **Sibren**, **Stan**, **Sam**, **Saqar**, **Joey**, **Varsha**, **Ana**, and **Dima** - you all contributed to making the group more diverse, welcoming, and fun. And also the other colleagues from the department: **José**, **Bowy**, **Julien**, **Pietro**, **Klaiv**, and **Toshi**. I will always remember the two-hour lunch breaks and all the beers we had sitting at the corner table at Doerak.

Big thanks to my bionanoscience and neuroscience collaborators: **Anouk**, **Dimphna**, and **Geeske**. It was an enriching experience to work on a project so far from my comfort zone; I learned a lot, and hopefully you did too.

People always say that when you live abroad, your friends become your family, and that couldn't be truer. I want to thank all my **friends who became family in Delft and Rotterdam**, many from the Brazilian community, whose warmth made the Netherlands truly feel like home.

Também queria agradecer **minha família**, que levo no coração e sinto saudade todos os dias. **Pai** e **Mãe**, seria impossível chegar até aqui sem o suporte de vocês, que sempre acreditaram em mim e me deram tudo o que precisei para poder focar nos estudos. A escolha de morar fora é sempre difícil, mas é extremamente confortante saber que toda vez que vou de férias pro Brasil tenho tantas pessoas que amo me esperando. Também

foi inestimável ter parte da família morando no mesmo continente: **Tio André**, **Tia Daiane**, **Sofia** e **Giovanni**, obrigado pelas férias que passamos juntos e por estar sempre presentes.

Meus amigos de Pelotas são amigos que levarei para a vida toda, que, ajudaram a formar minha personalidade e minha visão de mundo. Queria citar em especial o **Bruno**, que quando eu vim morar em Delft estava começando o mestrado em Haia. Já ter um amigo aqui quando cheguei foi um privilégio que não todos têm, e provavelmente eu teria desistido se não fosse pela tua ajuda.

Por fim, eu gostaria de agradecer ao amor da minha vida, **Paola**, por ser minha eterna parceira e amiga. É difícil colocar em palavras o quanto és importante pra mim. Obrigado por tudo - por me apoiar antes e durante essa jornada, por ter mudado radicalmente de vida e vindo pra outro país comigo, por estar junto em todos os momentos, bons e ruins. Nada parece incerto no futuro sabendo que estaremos juntos. Te amo muito.

CURRICULUM VITÆ

Vinicius FONSECA HERNANDES

2021–2025	**Ph.D**, Physics Delft University of Technology, Delft, The Netherlands Thesis: Bridging simulation and experiment in nanoscience with AI Promotor: Ronald Hanson Co-Promotor: Eliska Greplova
2020–2021	**Master of Science**, Physics Universidade Federal de Pelotas, Pelotas, Brazil Thesis: Applied machine learning to phase classification of soft matter systems Advisor: Jose Rafael Bordin
2016–2019	**Bachelor of Science**, Physics Universidade Federal de Pelotas, Pelotas, Brazil
30-12-1096	Born in Pelotas, Brazil

LIST OF PUBLICATIONS

10. **V. Hernandes**, T. Spriggs, S. Khaleefah, E. Greplova, *Adiabatic Fine-Tuning of Neural Quantum States Enables Detection of Phase Transitions in Weight Space*, arXiv:2503.17140 (2025).

9. V. Gualtieri, C. Renshaw-Whitman, **V. Hernandes**, E. Greplova, *QDsim: A user-friendly toolbox for simulating large-scale quantum dot devices*, SciPost Physics Codebases **046** (2025).

8. **V. Hernandes**, A. M. Heuvelmans, V. Gualtieri, D. H. Meijer, G. M. Woerden, *autoMEA: Machine learning-based burst detection for multi-electrode array datasets*, Frontiers in Neuroscience **18**, 1446578 (2024).

7. **V. Hernandes**, E. Greplova, *Exploring biological neuronal correlations with quantum generative models*, Cell Reports Physical Science **6**, 8 (2025).

6. **V. Hernandes**, E. Greplova, *Modeling Neuronal Activity with Quantum Generative Adversarial Networks*, 2023 IEEE International Conference on Quantum Computing and Engineering (QCE)(2023).

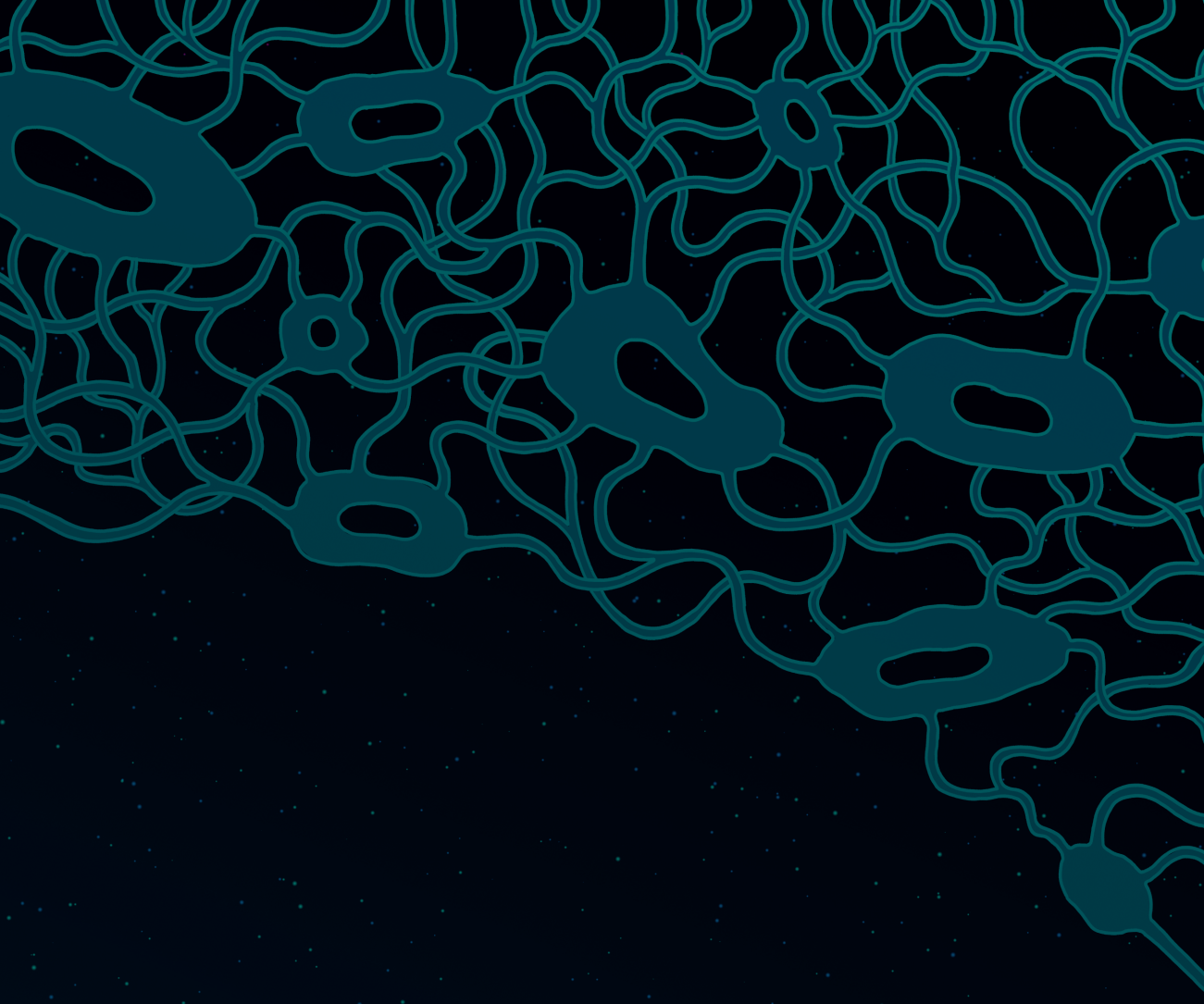
5. A. R. das Neves Stigger, **V. F. Hernandes**, M. M. Ferrer, M. L. Moreira, *Optical and electrical features of calcium molybdate scheelite solar cells*, New Journal of Chemistry **47**(26), 12458–12467 (2023).

4. **V. F. Hernandes**, M. S. Marques, J. R. Bordin, *Phase classification using neural networks: application to supercooled, polymorphic core-softened mixtures*, Journal of Physics: Condensed Matter **34**(2), 024002 (2021).

3. D. S. Cardoso, **V. F. Hernandes**, T. P. O. Nogueira, J. R. Bordin, *Structural behavior of a two length scale core-softened fluid in two dimensions*, Physica A: Statistical Mechanics and its Applications **566**, 125628 (2021).

2. M. S. Marques, **V. F. Hernandes**, J. R. Bordin, *Core-softened water–alcohol mixtures: the solute-size effects*, Physical Chemistry Chemical Physics **23**(30), 16213–16223 (2021).

1. M. S. Marques, **V. F. Hernandes**, E. Lomba, J. R. Bordin, *Competing interactions near the liquid-liquid phase transition of core-softened water/methanol mixtures*, Journal of Molecular Liquids **320**, 114420 (2020).



TU Delft

ISBN 978-94-6496-492-9