A Deep Learning Method for Fast Generation of Degenerate High-Dimensional Conductance-Based Populations from Neuronal Spike Times

Auteur : Brandoit, Julien
Promoteur(s) : Drion, Guillaume
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil biomédical, à finalité spécialisée
Année académique : 2024-2025
URI/URL : https://github.com/julienbrandoit/Spike2Pop---Bridging-Experimental-Neuroscience-and-Computationa
https://github.com/julienbrandoit/master-thesis-automatic-degenerate-cbm; http://hdl.handle.net/2268.2/23306

University of Liège
Faculty of Applied Sciences

A Deep Learning Method for Fast Generation of Degenerate High-Dimensional Conductance-Based Populations from Neuronal Spike Times

Master's thesis completed in order to obtain the degree of

Master of Science in Biomedical Engineering

by

BRANDOIT Julien

Academic supervisor:
Guillaume DRION

Co-supervisor:
Arthur FYON

Academic Year 2024–2025

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisor, Guillaume Drion, and my co-supervisor, Arthur Fyon. They provided me with just the right balance of support and independence, allowing me to dive into the fascinating — and somewhat new to me — world of computational neuroscience. Their guidance made the entire journey both enriching and enjoyable. They were always available and quick to respond whenever I had questions. A special thanks to Arthur for his time, his constructive feedback (both positive and negative, yet always fast), and his countless detailed comments throughout the writing of the (quite lengthy) manuscript. Thanks to both of them, my curiosity and enthusiasm for research have only grown stronger!

I also thank Loris Mendolia for our discussions on the perspective about neuromorphic chips and his feedback on the related section of the manuscript.

I am incredibly thankful to my parents, who have always encouraged me in my studies and personal growth — and I can happily say it paid off!

To Anaïs, who shares my daily life, thank you for your patience and support during the many evenings I spent in deep conversation with my computer rather than with you.

Thanks to Lindford, for the steady friendship throughout, even from outside the academic bubble. And of course, to my friends with whom I shared the university days — Fanny, Ahmed, Lucas, Clémence, and Linpha. We laughed a lot, and that made all the difference.

As promised, I'd like to end these acknowledgments with our philosophy:

"*If there's no risk, there's no fun.*" — A universal truth, according to *us*.

Abstract

Motivation and Objectives

Understanding the human brain complexity is a central goal in neuroscience. A key concept is *neuronal degeneracy* — the ability of structurally or functionally distinct neural configurations (e.g., ion channel densities) to produce similar outputs. Ion channels are transmembrane proteins that regulate the flow of specific ions across the neuron membrane, directly influencing electrical activity. While degeneracy contributes to robustness and adaptability, it complicates efforts to infer specific biophysical parameters from observed activity.

Conductance-based models (CBMs) are widely used due to their biological interpretability. In this context, conductance refers to how easily ions flow through a given ion channel, determining the strength of the resulting electrical current. However, determining the maximal conductance values that reproduce experimental data is challenging and often relies on trial-and-error. Furthermore, degeneracy means multiple parameter sets can yield similar behaviors, limiting interpretability and practical use by experimentalists.

This thesis addresses these challenges with a novel pipeline that combines the Dynamic Input Conductances (DICs) theory with deep learning. The objective is to generate degenerate CBM populations that replicate target neuronal activity — defined through spike times — directly from experimental recordings. This approach strengthens the connection between the experimental world and the modeling one, enabling systematic studies of degeneracy.

Key Contributions[1]

- **Deep Learning Pipeline**: A supervised model is trained to map spike time sequences to DIC values, which guide the generation of degenerate CBMs. Validated on synthetic data from two neuron types: the stomatogastric ganglion and the dopaminergic neuron.

- **Iterative Compensation Algorithm**: An improved method for generating degenerate CBM populations by precisely targeting DIC values at threshold voltage with minimal computational cost.

- **Reachability Analysis in DIC Space**: A theoretical contribution identifying which DIC regions correspond to biologically plausible CBMs (i.e., those with positive conductances), providing a heuristic for conductance selection and insight into how ion channel variations affect activity.

- **Open-Source Software Tool**: A user-friendly application that allows experimentalists to generate and validate CBM populations without needing expertise in programming or machine learning. It supports multiple CBMs and includes a graphical interface.

Results

The results are very satisfactory with a tool that demonstrates robustness to noise, generated populations that faithfully reproduce the target activity, and validation on two different CBMs, thus highlighting the generalizability of the method. The developed application truly aims to integrate the method into a laboratory context and can be executed quickly on any personal computer.

The perspectives touch on both biology, computational neuroscience, and neuromorphic systems.

[1] This thesis repository: https://github.com/julienbrandoit/master-thesis-automatic-degenerate-cbm
The developed software repository:
https://github.com/julienbrandoit/Spike2Pop---Bridging-Experimental-Neuroscience-and-Computational-Modeling

Contents

Figures

Tables

Summary Boxes

Chapter 1

Introduction

1.1 Motivation

The human brain remains one of the most enigmatic and complex systems in nature. Understanding its mechanisms is fundamental to unraveling cognition, advancing treatments for neurological disorders, and driving innovation in artificial intelligence. However, one of the central challenges in neurosciences is decoding how diverse neural components interact to produce stable and adaptable behavior.

Neuronal degeneracy — the property whereby distinct, structurally or functionally diverse neural parameters, such as membrane composition or synaptic strengths, produce similar activity — lies at the core of brain organization. Unlike redundancy, where identical components replicate a function, degeneracy allows different neural circuits, cellular mechanisms, or synaptic configurations to support the same function. This diversity enhances the brain robustness to developmental variability, injury, or disease by offering multiple pathways to perform essential functions.

In short, the concept of degeneracy can be summed up by the idea that "*there exist multiple solutions leading to similar or nearly identical behaviors*" (Goaillard and Marder, 2021). While this property provides resilience and adaptability, it also presents a significant challenge: when non-identical configurations produce similar spiking patterns or cognitive outputs, identifying the specific underlying biophysical parameters responsible for these behaviors becomes a nontrivial task, especially in computational work where analyses are often quantitative.

Mathematical and numerical tools play a central role in *understanding* or *reproducing* neural mechanisms, or both — whether in biology, medicine, or neuromorphic engineering. Grouped under the term "*computational neuroscience*", these tools are powerful instruments that open the way to *in silico*[1] experiments in neurosciences.

Among computational neuroscience models offering high biological interpretability, *conductance-based models* (CBMs) occupy a major historical and technical position (Almog and Korngreen, 2016). Indeed, from the pioneering work of Hodgkin and Huxley, 1952 to more recent models such as those by Yu and Canavier, 2015 or Taylor et al., 2009, CBMs have demonstrated their ability to faithfully represent biological observations while maintaining a direct interpretation linked to biology through their parameters. In particular, *maximal conductances* are especially meaningful, as they are directly related to underlying biological structures such as the type, density, and distribution of ion channels in the neuronal membrane.

Despite their potential, biophysical computational tools often face reluctance from experimentalists, even though their intrinsic structures are based on biological mechanisms. This limits the practical impact of computational neuroscience work and hinders scientific progress.

The sources of this reluctance are twofold, but not independent. The first is the difficulty of reproducing laboratory observations using *in silico* models. While the models themselves are capable, determining the

[1] *In silico* refers to experiments or models conducted via computer simulation or computational methods, as opposed to those conducted in a laboratory or in living organisms (which are referred to as *in vitro* or *in vivo*, respectively).

values of the parameters required for faithful reproduction is a long, time-consuming, and tedious process that requires a certain amount of technical expertise. This process is often carried out through trial and error (e.g., Nadim et al., 1995; Traub et al., 1991).

Moreover, it appears that between the model of Hodgkin and Huxley, 1952 and those of Taylor et al., 2009; Yu and Canavier, 2015 or others, the minimum number of parameters to be determined by the experimentalists grows drastically up to several hundred parameters in the latest models. This observation accentuates the gap between the ease of use for the experimentalists and the fidelity of the biological representation.

While the nature and number of parameters vary across models, experimentalists must carefully select and adjust them to align simulations with experimental data. These adjustments, influenced by the model structure and the inherent constraints it imposes, are crucial in ensuring biological accuracy. However, experimentalists cannot measure all neuronal parameters directly — some are determined through measurement, while others must be approximated, sometimes with great difficulty, adding a layer of error and complexity.

The second reason for the resistance shown by experimentalists is closely linked to the notion of degeneracy. Indeed, in addition to the need to determine parameter values to reproduce observations, the experimentalists must be able to identify a set — ideally the complete set — of these parameters when they wish to study the impact of external influences on the activity. This ensures that the results are not only valid for a particular subset of conditions, thus ensuring a better extrapolation towards a global biological understanding. Moreover, there is considerable evidence from computational neuroscience, including works by Marder and Taylor, 2011 or Golowasch et al., 2002, that representation by model sets is really necessary, and that the average model is often a bad representation.

This highlights the importance of developing tools in computational neuroscience that not only study neuronal phenomena in the context of degeneracy but also bridge the gap between experimentalists' practical observations and the complexity of detailed models. These tools should facilitate the translation of *in vivo* or *in vitro* data into *in silico* models that maintain high biological interpretability, making it easier for experimentalists to work with and validate these models.

In this perspective, and in addition to improving the representational fidelity of biological behaviors, a number of recent works in computational neuroscience are working to narrow this gap for experimentalists. Among them, several mathematical analyses of CBMs aim to abstract and mitigate the problem of high dimensionality, notably by leveraging bifurcation theory. These mathematical tools significantly enhance the study of CBMs that are inherently *intractable* — meaning that, without explicit simulation, we cannot *a priori* easily predict the activity associated with a given set of parameters — by making these models more manageable and easier to analyze, thus reducing the computational resources typically required.

Among recent approaches to simplifying the analysis of CBMs, the Dynamic Input Conductances (DICs) approach proposed by Drion et al., 2015 stands out for its simplicity and effectiveness. It consists in condensing, by means of only a few voltage-dependent curves, the combined effect of all the parameters on different timescales. This dimensional reduction enables experimental observations to be compared more directly with models, at the expense of a certain loss of direct biological interpretability. At the same time, it provides a way to encapsulate degeneracy by grouping together different parametric configurations that give rise to similar neuronal behaviors.

Meanwhile, the recent publication of Fyon et al., 2024 sheds new light on the mechanisms underlying neuronal degeneracy. It highlights two key mechanisms that explain how different parametric configurations can lead to equivalent behaviors. The authors propose an efficient method for generating degenerate populations by sampling ionic conductance distributions while controlling these mechanisms. This approach provides a systematic and faster alternative to random exploration, allowing for a more controlled and efficient study of parametric combinations that are consistent with similar neuronal activities, thus facilitating quantitative analyses of degeneracy in CBMs.

Although the generation of degenerate populations is now possible by sampling ionic conductance distributions, the practical use of these tools remains limited by the absence of a systematic framework for directly linking recorded neuronal activities during experiments to the underlying inputs needed to generate those populations. This thesis directly addresses this limitation by developing a tool that maps recorded neuronal activity to a population of conductance-based models that replicate such activity.

The objective of this tool is to create a direct and efficient connection between experimental data and the parameters of CBMs. By doing so, it provides a way to identify multiple conductance configurations that produce a target similar neuronal activity, thus overcoming the challenge of degeneracy in model interpretation.

The proposed solution is based on a deep learning architecture, designed to operate under experimental conditions where only spike times sequences are available — a constraint that ensures the tool broad applicability across various experimental setups. At the same time, it is flexible enough to be adapted to different CBMs, ensuring compatibility with a variety of biologically plausible models.

In addition to proving the concept, this thesis aims to deliver an open-source, intuitive, and optimized software that bridges the gap between experimental neurosciences and computational modeling. By providing a unified framework for studying neuronal degeneracy and simplifying the use of CBMs, this work makes computational neuroscience more accessible to experimental researchers, thus facilitating the integration of modeling tools into biological research.

1.2 Scientific Contributions

This section summarizes the scientific contributions of this work, which serves two main purposes. First, as a Master Thesis, it aims to demonstrate the ability to independently manage a large project, showcasing a deep understanding of the subject matter. This is reflected in the detailed background sections, which are more extensive than what readers might find in a typical research paper. Second, this work also aims to contribute new findings to the field. The contributions are divided into two main areas:

- **Technical Contributions**: These involve creating new tools and methods. Whether through new designs, combinations, or applications in neuroscience, these contributions add practical value and improve methodologies.

- **Theoretical Contributions**: These provide new ideas or answers that deepen our understanding of neuroscience, computational neuroscience, or neuromorphic engineering.

For a concise summary of the research question and the main contributions of this work, please refer to Box I.

1.2.1 Technical Contributions

The major contribution of this work is the development of a pipeline that combines tools from the theory of Dynamic Input Conductances (DICs) with a deep learning architecture. This combination provides an effective and robust data-driven solution for the automatic generation of degenerate populations of Conductance-Based Models (CBMs) with target activity. As proof of the method generality, it is applied to two different CBMs: a stomatogastric ganglion (STG) neuron model and a dopaminergic (DA) neuron model.

The pipeline robustness is evaluated under experimental conditions, such as noise in the data, demonstrating its practical applicability. Quantitative analyses of data efficiency and hyperparameter impacts are also provided, resulting in a pipeline that is efficient, fast in inference, and has a small number of parameters. This makes it more data-efficient compared to similar approaches in the literature.

A second technical contribution is the improvement of the degenerate population generation method initially proposed by Fyon et al., 2024. The original method had significant residuals, but the proposed solution, namely *the iterative compensation algorithm*, allows for precise enforcement of the DIC values at the threshold potential, with minimal additional computational cost, which is an important step in the generation procedure.

Additionally, this work provides an open-source repository with all the Python code developed for this project.[2] This includes a translation of many elements originally developed in Julia by Arthur Fyon[3]. The implementation leverages efficient numerical vectorized computation and multi-CPU parallelization.

Finally, to fully address the broader problem of integrating computational neuroscience tools into the daily work of experimentalists, a user-friendly *software package* is developed[4]. This package does not require expertise in deep learning or programming. It provides an interface to the pipeline, allowing users to easily generate CBM populations from laboratory recordings through a simple graphical interface. Experimentalists can select the type of recorded neuron and the size of the population to generate, and get simulation results and conductances distributions.

1.2.2 Theoretical Contributions

Initially developed as a technical solution, this work explores *reachability* in the DICs space and proposes results and a numerical method for constructing reachability maps for any CBM. To our knowledge, this is the first time this question has been addressed in the literature. Reachability investigates which points in the DICs space lead to biologically plausible populations, i.e., those with positive maximal conductance values, based on the set of compensated (modulated) conductances.

In this work, reachability is used to build a heuristic for selecting conductances to compensate during population generation. However, we believe these maps can be extended and used for theoretical understanding of biology, particularly to comprehend what can, and especially what cannot, explain the transition from one neuronal activity to another through variations in maximal conductances during cellular perturbations (neuromodulators, diseases, etc.).

1.3 Document Organization

This thesis is organized into several chapters, each addressing specific aspects of the research and methodology. The structure is designed to provide a clear and logical flow from background information to the presentation of results and future perspectives.

Chapter 1: Introduction The introduction provides the context and motivation for the research, discussing the complexity of the human brain and the challenges in understanding its mechanisms. It introduces the concept of neuronal degeneracy and outlines the objectives and contributions of the thesis.

Chapter 2: Background — Neurosciences and Computational Neuroscience This chapter provides foundational knowledge in neuroscience and computational neuroscience. It covers the basics of neurons, neuronal degeneracy, and conductance-based models. Specific models, such as the stomatogastric ganglion neuron model and the dopaminergic neuron model, are discussed in detail. The chapter also explores the theory of Dynamic Input Conductances and an efficient algorithm for generating degenerate populations.

[2]This thesis repository: https://github.com/julienbrandoit/master-thesis-automatic-degenerate-cbm.

[3]Arthur Fyon's repository: https://github.com/arthur-fyon/CORR_2024.

[4]The "*Spike2Pop*" application repository:
https://github.com/julienbrandoit/Spike2Pop---Bridging-Experimental-Neuroscience-and-Computational-Modeling

Chapter 3: Background — Supervised Learning and Deep Learning This chapter delves into the fundamentals of supervised learning and deep learning. It explains the general framework of supervised learning, formulates the specific problem addressed in this work, and discusses deep learning techniques, including architectural building blocks and transfer learning methods.

Chapter 4: Literature Review — Automated Generation of Conductance-Based Models with Targeted Activity This chapter reviews existing literature on the automated generation of conductance-based models with targeted activity. It provides a comprehensive overview of current methods and approaches in the field. It clearly positions this thesis and highlights its novelty.

Chapter 5: Development and Methodology This chapter details the development and methodology used in this research. It includes preliminary considerations, the iterative compensation algorithm, reachability and its limits, dataset generation, and the model architecture and training strategy. Additionally, it discusses the transfer of the model to the dopaminergic neuron model.

Chapter 6: Results and Discussion This chapter presents the results of the research and discusses their implications. It covers hyperparameter tuning, data efficiency analysis, final model evaluation, robustness analysis, and the results of transferring the model to the dopaminergic neuron model.

Chapter 7: Development of an Open-Source Software for Computational Neuroscience This chapter describes the open-source software package designed to facilitate the integration of computational neuroscience tools into experimental workflows. It discusses the features and functionalities of the software, emphasizing its user-friendly interface and practical applications.

Chapter 8: Limitations and Perspectives The final chapter explores the limitations of the research, potential future directions, and the broader applications of the pipeline. It discusses how the developed tools and methods can be applied in laboratory settings and neuromorphic system modulation.

Appendices The appendices provide additional details and supporting information. Readers are redirected to the outline of the Appendices for more details.

This organizational structure ensures that the thesis is comprehensive, logically sequenced, and accessible to readers, providing a clear path from background information to methodological development, results, and future perspectives.

Box I – Summary of Research Contributions

Research Question

The central research question addressed in this thesis is: "*How can we generate degenerate populations of conductance-based models (CBMs) with a target activity using only recordings of spike times?*" This question is motivated by the need to bridge the gap between experimental data, the reality of experimentalists' work, and computational modeling, facilitating the study of neuronal degeneracy.

Main Contributions

This work introduces several key contributions to the field of computational neuroscience:

1. **Pipeline Development:** Development of a robust pipeline that combines the theory of Dynamic Input Conductances (DICs) with a deep learning architecture to generate degenerate CBM populations. This pipeline is validated using synthetic data and is adaptable to different CBMs.

2. **Iterative Compensation Algorithm:** Improvement of existing methods for generating degenerate populations by introducing an iterative compensation algorithm. This algorithm enhances the precision of targeting specific DIC values with minimal computational overhead.

3. **Open-Source Software:** Creation of an open-source software package that provides a user-friendly interface for experimentalists to generate and validate CBM populations. This software does not require expertise in deep learning or programming.

4. **Theoretical Insights:** Exploration of reachability in the DICs space, providing a heuristic for selecting conductances to compensate during population generation. This contributes to the theoretical understanding of neuronal activity and degeneracy.

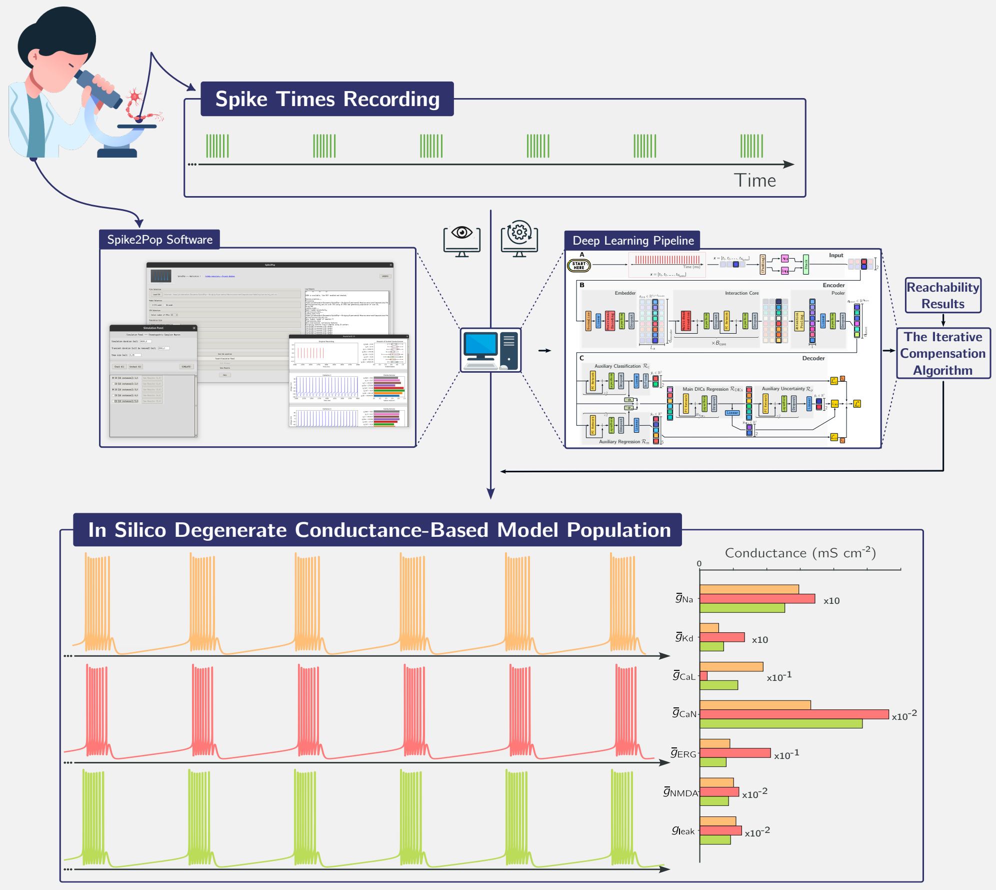

Spike Times Recording
Time
Spike2Pop Software
Deep Learning Pipeline
Reachability Results
The Iterative Compensation Algorithm
In Silico Degenerate Conductance-Based Model Population
Conductance (mS cm^{-2})
0
$\bar{g}_{Na}$ x10
$\bar{g}_{Kd}$ x10
$\bar{g}_{CaL}$ x10^{-1}
$\bar{g}_{CaN}$ x10^{-2}
$\bar{g}_{ERG}$ x10^{-1}
$\bar{g}_{NMDA}$ x10^{-2}
g_{leak} x10^{-2}

Figure 1.1: Pipeline for generating degenerate conductance-based models from neuronal spike times recording.

These contributions not only advance our understanding of neuronal degeneracy but also provide practical tools for experimentalists, fostering interdisciplinary collaboration and innovation in neurosciences research.

Chapter 2

Background — Neurosciences and Computational Neuroscience

To effectively model and analyze neuronal behavior, we need to connect experimental observations with mathematical models. This chapter provides the essential biological and computational background necessary to understand the methods and models used in this thesis. It covers key concepts in neurosciences, introduces Conductance-Based Models (CBMs) we use, and explains how these models help us study neuronal dynamics, focusing particularly on degeneracy.

Structure of the Chapter

The first section, **2.1 Biological Background**, provides a foundational understanding of how neurons process the information. We'll cover the structure of neurons, how they generate electrical signals (action potentials), and the concept of neuronal degeneracy. A summary of key biological concepts is provided in Box II and can be used by readers less familiar with biology throughout this document.

The second section, **2.2 Conductance-Based Models** (CBMs), introduces mathematical representations of neurons. These models simulate neuronal activity by representing the electrical behavior of neurons through components such as ion channels and describing the neuronal membrane as a nonlinear RC electrical circuit. We'll examine two specific models: one for neurons in the stomatogastric ganglion of crustaceans (STG) and another for dopaminergic (DA) neurons. These models help us understand how neurons generate different types of activity, such as regular spiking or bursting. The structure and key components of CBMs are explained in Box III.

The third section, **2.3 The Theory of Dynamic Input Conductances**, introduces a powerful mathematical tool called Dynamic Input Conductances (DICs). DICs allow us to simplify complex models by breaking down their behavior into different timescales. This simplifies our qualitative understanding of model activity by reducing complexity and helps us grasp degeneracy by demonstrating how different parameter sets can yield similar activity. The key ideas behind DICs are summarized in Box IV.

The final section, **2.4 An Efficient Algorithm to Generate Degenerate Populations**, describes a method for creating populations of neuron models that all show similar activity but have different parameters in a computationally efficient way. We'll discuss how this method works and how it can be improved. The main steps of the algorithm are outlined in Box V.

By the end of this chapter, the reader should have a clear understanding of the biological and computational foundations necessary to follow the methods, results, and applications discussed in the subsequent chapters of this thesis.

2.1 Biological Background

To bridge the gap between experimental observations and computational models, understanding the fundamental principles of neuronal activity is essential. This section provides the reader with all the essential biological background needed to understand the main messages of this thesis.

2.1.1 The Basics of Neurons

Neurons are the fundamental cellular units underlying cognition and neural processing in the nervous system. They are specialized cells responsible for processing and transmitting information through electrical and chemical signals, enabling rapid communication within the nervous system. The brain, spinal cord, and all the nerves are composed of neurons, forming a vast and complex network involved in many physiological processes such as movement, decision-making, and behavior regulation. [1]

As complex functional units, neurons are crucial to understand biological systems, disease mechanisms, and the development of efficient neuromorphic systems. Indeed, the nervous system, considered as a computational unit, stands out for its remarkable energy efficiency. Although it is one of the most energy-consuming organs in the human body (Wang et al., 2010), it is still capable of learning and generalizing even in *low-data* contexts while controlling a wide variety of tasks from a single central unit. These properties make the brain and neurons a source of inspiration for the development of new technologies, grouped under the field of "*neuromorphic engineering*."

The nervous system has a complex structure and involves many types of cells beyond neurons. However, this section focuses solely on the isolated neuron while acknowledging that neurons function within networks to produce complex functions.

The Anatomy of Neurons

Neurons are, first and foremost, *cells* and, as such, can be described as a set of biological structures organized into *organelles* or other components, composed of biomolecules such as proteins and nucleic acids, immersed in the *cytosol* — the aqueous solution of the *cytoplasm*. This intracellular domain is separated from its external environment, referred to as the extracellular domain, by the *cell membrane*, a boundary that electrically isolates the two environments while selectively regulating chemical exchanges.

Two features distinguish neurons from other cell types:

- **Electrical excitability**: Unlike typical cells, neurons can generate and propagate electrical signals called *action potentials*. This property relies on the presence of **voltage-gated ion channels**, which enable the rapid movement of ions across the cell membrane.

- **Synaptic specialization**: Neurons communicate with each other through specialized structures called *synapses*. These allow signal transmission either in a chemical form, via neurotransmitters, or in an electrical form through gap junctions. However, this important aspect of communication between neurons is beyond the scope of this work.

Figure 2.1 presents a general schematic representation of a neuron structure, alongside an electron microscopy image of a neuron. Photomicrographs can be found in Chapter 1 of Hammond, 2024 (e.g.,

[1] This part of my work is partially based on information found in Hammond, 2024. To avoid overloading the text, I make a single reference here. In particular, Chapters 1 ('*Neurons*'), 2 ('*Neuron-glial cell cooperation*'), 3 ('*Ionic gradients, membrane potential and ionic currents*'), 4 ('*The voltage-gated channels of* Na^+ *action potentials*'), 5 ('*The voltage-gated channels of* Ca^{2+} *action potentials: generalization*'), and 18 ('*Firing patterns of neurons*'); as well as, to a lesser extent, Chapters 6 ('*The chemical synapses*') and 7 ('*Neurotransmitter release*'). Other references will be provided as needed throughout the text, but Hammond, 2024 is at the core of this **Biological Background** section of my work.

Fig. 1.1 and 1.3). The exact structure, particularly the dendritic tree and the axon, varies significantly between different types of neurons and even within the same neuron family.

Neurons are not merely homogeneous cells but highly specialized structures in which each region plays a distinct role, ensuring an optimized functional organization. This regionalization is essential for their ability to efficiently process, transmit, and modulate information. It affects not only cellular metabolism but also the reception, conduction, and transmission of electrical signals, as well as the secretory function involved in synaptic communication.

The cell membrane Neurons are immersed in an ionic environment, with the extracellular fluid containing various ions, including sodium (Na^+), potassium (K^+), calcium (Ca^{2+}), chloride (Cl^-), and others. The neuron *plasma membrane* is a phospholipid bilayer embedded with various proteins, including *ion channels*, *pumps*, and *transporters*. Importantly, the phospholipid bilayer itself acts as an *electrical insulator*, preventing the free flow of ions across the membrane. This insulating property is essential for maintaining distinct ionic concentrations on either side of the membrane. The embedded proteins regulate the selective movement of ions, thereby establishing and maintaining the ionic gradients necessary for neuronal function. These gradients are critical for generating electrical signals, such as the *action potential*.

The soma (or cell body) forms the central region of the neuron, where the *nucleus* and most of the organelles involved in protein synthesis and cellular metabolism are located. It is within the soma that signals received by the dendrites are integrated, and an action potential can be triggered in response to sufficient stimulation or the cell intrinsic dynamics.

The dendrites are branched extensions emerging from the soma that serve as the primary sites for receiving signals from other neurons or external stimuli. They contain numerous *dendritic spines*, which increase synaptic contact surface and play a role in neuronal plasticity.

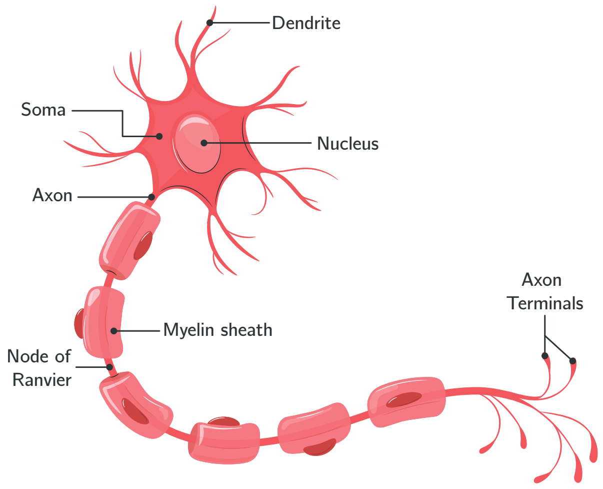
The axon is an elongated structure that transmits electrical signals from the soma to other neurons, muscles, or glands. Its length can range from a few micrometers to over a meter. At its end, it branches into *axon terminals*, which establish synapses with target cells.

The myelin sheath In most neurons, the axon is covered by a *myelin sheath*, consisting of concentric layers of lipid membrane produced by glial cells. This structure significantly enhances the speed of action potential conduction through a propagation mode known as *saltatory conduction*. The myelin sheath is interrupted at regular intervals by regions devoid of myelin, called *nodes of Ranvier*. These nodes play a role in allowing action potentials to regenerate at each interval, thereby reducing signal dissipation.

The synapses are specialized structures through which neurons communicate with each other or with other target cells, such as muscle or glandular cells. They act as the connection points that allow information to transfer from the axon terminals of one neuron to the dendrites of the next.

A neuron can be viewed as an input-output system, where synaptic inputs received by dendrites are integrated within the soma, processed into an all-or-none action potential, and transmitted as an electrical signal along the axon to release neurotransmitters at the axon terminals, influencing the activity of downstream cells.

In this work, we approach neurons through models that focus on their electrical activity, which is described in the following of this section. The general anatomical description of a neuron presented above represents a simplification of current knowledge in neurocytology. However, it provides the foundation for the abstraction used by *Conductance-Based Models* (CBMs), introduced in Section 2.2, where a neuron is modeled as a unit enclosed by a cell membrane that separates an intracellular and an extracellular environment with distinct ionic compositions. This membrane is not a passive barrier but a dynamic structure containing a set of specific proteins — including ion channels — that confer selective permeability to ions.



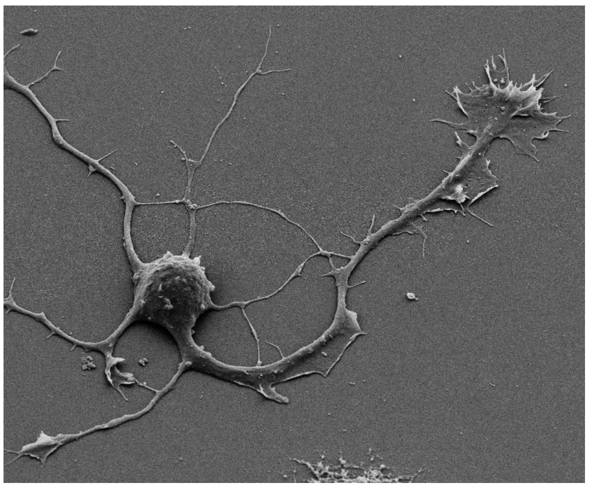


Figure 2.1: General neuron anatomy. *Left:* Schematic representation of a general neuron structure — *Adapted from*[a]. *Right:* Electron microscopy image of a neuron — *Taken from*[b].

[a]Illustration by Storyset, "Nerve Cell Concept Illustration," Freepik, accessed March 9, 2025, https://www.freepik.com/free-vector/nerve-cell-concept-illustration_42106424.htm
[b]Image from the University of North Carolina, accessed March 9, 2025, https://www.med.unc.edu/corefacilities/wp-content/uploads/sites/943/2021/11/image001.png

Thus, the neuron can integrate signals from its environment through synaptic inputs and produce an electrical response, which can then be transmitted to other neurons or effector cells.

Neurons are Excitable Cells

Neurons are excitable cells, meaning they have the ability to respond to external stimuli by generating and propagating electrical signals known as *action potentials*, but only when the stimulus is strong enough to exceed a certain threshold. Stimuli that fall below this threshold, known as subthreshold stimuli, do not trigger a response, which is an important characteristic of excitability. This excitability is essential for the proper functioning of the nervous system, as it enables neurons to transmit information in an *event-driven* manner, allowing for efficient communication with other cells. This event-based processing reduces energy consumption while maintaining the ability to respond to relevant stimuli.

The basis for neuronal excitability lies in the *membrane potential*, $V(t)$, which is the voltage difference across the neuronal membrane. This potential results from the difference in ion concentrations across the cell membrane. The movement of these ions across it is controlled by ion channels that open and close in response to changes in the membrane potential, allowing for the rapid changes in voltage necessary for signal transmission. A schematic representation of the membrane, showing the distribution of ions inside and outside the neuron as well as some of the key ion channels, is provided in Figure 2.2.

Resting potential of a neuron is the electrical potential difference across the cell membrane when the neuron is not actively transmitting a signal. This potential is typically negative, ranging from -60 to $-80\,\mathrm{mV}$ in most neurons. More specifically, the intracellular fluid has a higher concentration of K^+ ions and a lower concentration of Na^+ ions compared to the extracellular fluid, which is rich in Na^+ and Cl^-. The membrane selective permeability through ion channels, particularly to potassium ions, and the activity of the sodium-potassium pump, which maintains the concentration gradients, work together to stabilize the resting potential. The pump actively moves Na^+ out of the cell and K^+ into the cell, creating a separation of charges that results in a negative potential inside the neuron. More details on the driving force behind ionic movement across the membrane, in particular the concept of *electrochemical gradient*, will be detailed later in this section.

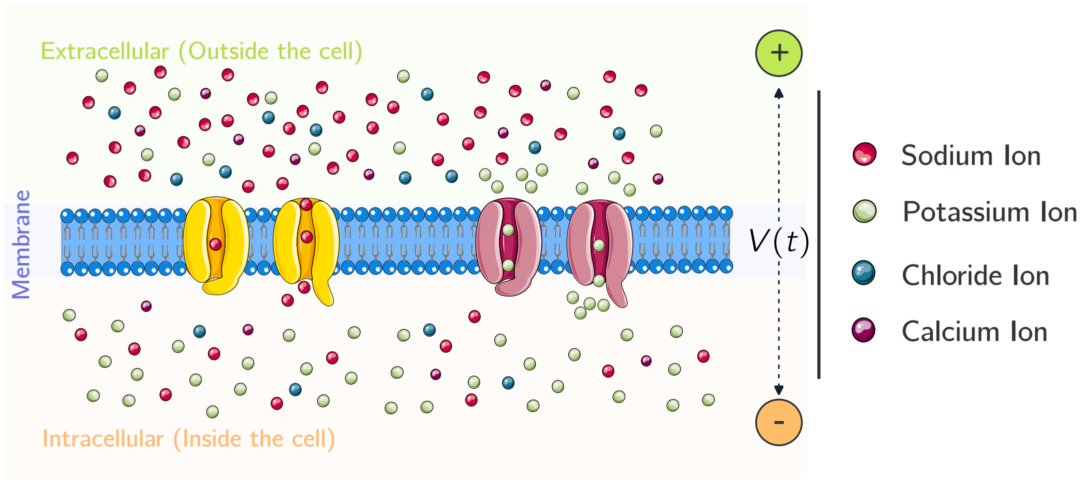

Extracellular (Outside the cell)
Membrane
Intracellular (Inside the cell)
+
$V(t)$
-
Sodium Ion
Potassium Ion
Chloride Ion
Calcium Ion

Figure 2.2: Schematic representation of a cell membrane with ion distribution and ion channels. Sodium (red), chloride (blue) and calcium (purple) ions are more concentrated in the extracellular space, while potassium (green) is more concentrated in the intracellular space. At rest, the intracellular side of the membrane is negative compared to the extracellular side due to the charge difference. Ion channels embedded in the membrane regulate the movement of these ions, contributing to the establishment and maintenance of the membrane potential $V(t)$.[a]

[a]Ion, channel and membrane icons are taken from Servier, licensed under CC-BY 4.0, https://smart.servier.com/.

Action potential An *action potential* is a rapid, highly reproducible, transient change in the membrane potential that propagates along the axon, allowing neurons to transmit electrical signals over long distances. It is a defining feature of neuronal excitability and occurs in response to a stimulus strong enough to depolarize the membrane to a certain threshold. Once this threshold is reached, voltage-gated ion channels open, enabling ions to flow across the membrane and triggering the action potential. The process can be broken down into several distinct phases:

1. **Depolarization**: When the threshold potential is reached, voltage-gated sodium channels open, allowing Na^+ ions to flow into the cell. This influx of positive ions causes the membrane potential to become less negative (i.e., depolarized).

2. **Repolarization**: As the membrane potential nears its peak, sodium channels close, and voltage-gated potassium channels, which open more slowly, begin to open. Potassium ions exit the cell, helping to restore the resting potential.

While sodium and potassium ions play the most prominent roles in the action potential, other ions, such as calcium and chloride, also contribute to the overall ionic balance and membrane dynamics.

During an action potential, the neuron enters a *refractory period*, which helps ensure the one-way propagation of the signal and limits the frequency of firing. The refractory period consists of two phases: the *absolute refractory period*, during which no new action potential can be generated regardless of stimulus strength, and the *relative refractory period*, during which a stronger-than-usual stimulus is required to initiate another action potential.

A schematic representation of the action potential and its various phases is shown in Figure 2.3A. The action potential follows an all-or-none principle: once the threshold is reached, a full action potential propagates along the axon without diminishing in amplitude.

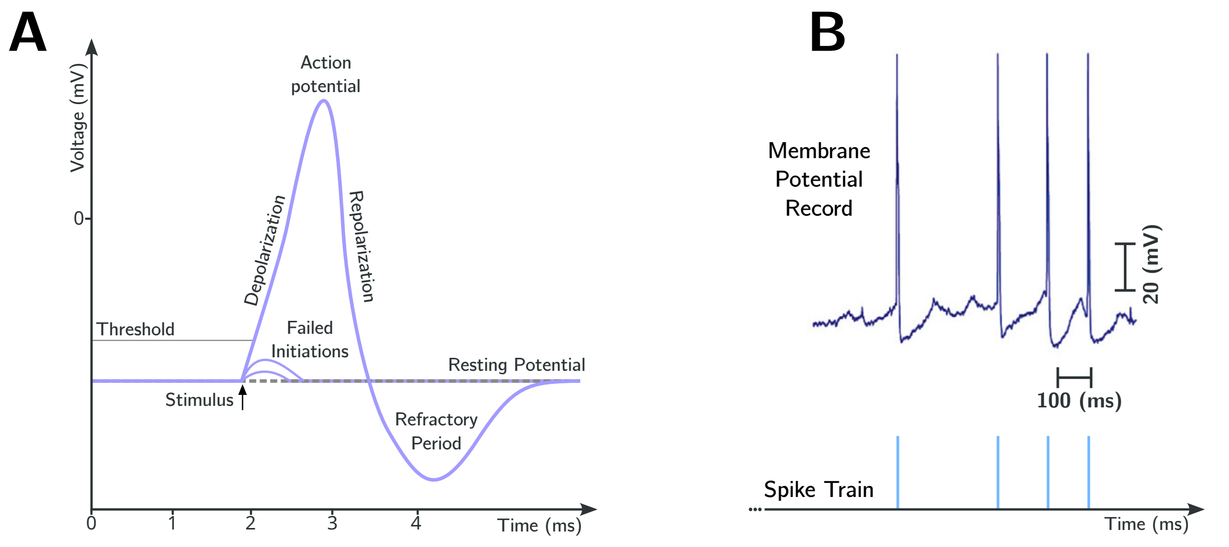


Figure 2.3: Schematic of the action potential and recorded neuronal activity. (A) The action potential is illustrated by the changes in the membrane potential over time, highlighting the phases of depolarization, repolarization, hyperpolarization, and return to resting potential — *Adapted from*[a]. (B) A typical recording of neuronal electrical activity, showing the action potential spikes sequences (*spike train*). The spike times are well-defined and less noisy compared to subthreshold fluctuations around the resting potential — *Modified from*[b].

[a]Image by Chris 73, *Action potential*, accessed March 9, 2025, via Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Action_potential.svg.
[b]Extracted from Hammond, 2024, Figure 18.6.

About the threshold potential A common misconception about the action potential is that the threshold is a fixed, precise voltage that must be reached for it to occur. In reality, the threshold is not a single, sharply defined value but rather a dynamic range of membrane potentials influenced by various factors. This behavior is more related to a resonator (Izhikevich, 2001) — when a stimulus is applied at the right moment, synchronizing with the neuron natural subthreshold oscillations, it can bring the membrane potential to the threshold more efficiently. Once the threshold is crossed, the neuron enters an all-or-nothing response, but if the threshold is not reached, no action potential will occur. This means that the timing and intensity of the stimulus play important roles in determining neuronal firing.

The Function of Neurons Through Their Electrical Activity

The excitable nature of neurons enables them to fulfill their function of **event-based communication**, meaning that communication occurs through the generation of action potentials triggered by specific stimulus conditions with sufficient intensity. The distribution of action potentials over time represents the neuron **firing activity**. Neurons can exhibit different activities, ranging from regular, rhythmic firing to irregular, bursting patterns, depending on their specific roles in the nervous system.

Figure 2.4 illustrates two examples of neuronal activity: **spiking activity** (A) and **bursting activity** (B). While spiking consists of isolated action potentials separated by relatively constant intervals, bursting involves a series of action potentials grouped into *bursts*, generated at a relatively high frequency, with each burst separated by a period of silence. These are just two examples, but in reality, there is a wide variety of other activity patterns exhibited by neurons.

Spike times The term *spike times* refers to the moments at which a neuron generates action potentials. In the context of a neuron firing activity, spike times and, especially their distribution, are crucial for encoding information. These times can vary depending on factors such as the intensity of the stimulus,

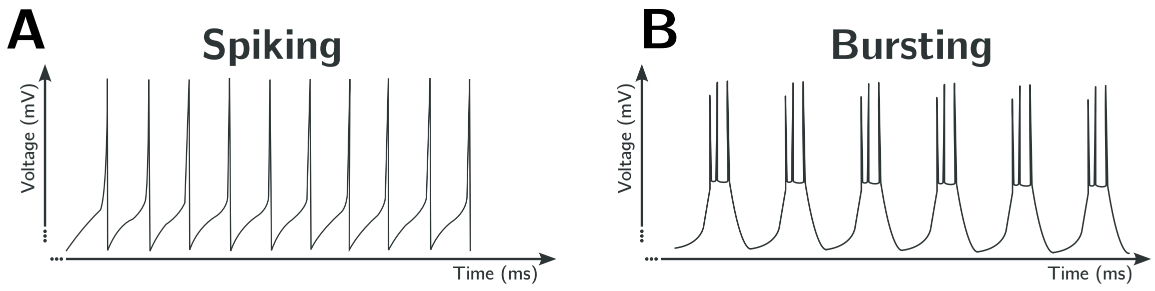


Figure 2.4: Examples of neuronal firing patterns. (A) **Spiking activity**: Isolated action potentials are generated at relatively constant intervals. (B) **Bursting activity**: A series of action potentials (here 3) grouped into bursts, generated at a high frequency with silent periods in between. These are just two examples, but neurons can exhibit a wide range of other firing patterns depending on their type and functional context — *Adapted from*[a].

[a]Turrigiano et al., 1995, Figure 1

the intrinsic properties of the neuron, and the type of synaptic inputs the neuron receives. Spiking patterns can be influenced by both the frequency of action potentials and the timing between them. This concept is particularly important in this work, since it represents the data that we assume available. More specifically, it is common in laboratory experiments to be able to record the full electrical activity of a neuron. However, the latter is noisy and the entire membrane voltage trace is difficult to use when dealing with high-dimensional networks. Action potentials, on the other hand, are well-marked events in neuronal activity, and the sequence of spike times can easily be recorded. This sequence is sometimes referred to as the '*spike train*' and is considered here to describe neuronal activity. Figure 2.3B shows a recording of the electrical activity of a neuron, and we can see that spike times are less noisy than subthreshold activity — i.e. activity around the resting potential.

Ion channels shape neuronal activity The electrical activity of neurons is shaped by the dynamics and distribution of *ion channels* embedded in the plasma membrane. Importantly, for a given neuronal type, the dynamic properties of ion channels remain relatively constant, such that it is primarily the density of the various channel types that determines the electrical activity. Their distribution can change over time through several mechanisms. Two illustrative examples are: (1) the constant turnover of membrane channels, in which the cell continuously recycles existing channels and synthesizes new ones; and (2) biochemical processes, such as *phosphorylation*, which can alter the *effective* distribution of ion channels by modifying their functional state or subcellular localization, without necessarily changing their absolute number. These mechanisms represent the intrinsic, cell-autonomous regulation of excitability. In parallel, neuronal activity can also be modulated by external stimuli, such as synaptic inputs or neuromodulatory signals, which dynamically influence the membrane potential and firing patterns.

The Nernst Potential and Ionic Gradients

Nernst Potential The Nernst potential, also known as the reverse potential, is the electrical potential that balances the concentration gradient of a specific ion across a membrane. In the simplest cases, for ion channels that are specific to a single ion, the reverse potential is determined by the Nernst equation:

$$E_{\text{ion}} = \frac{RT}{zF} \log\left(\frac{[\text{ion}]_{\text{extracellular}}}{[\text{ion}]_{\text{intracellular}}}\right), \tag{2.1}$$

where E_{ion} is the Nernst potential for the ion, R is the universal gas constant, F is the Faraday constant, z is the charge of the ion, T is the temperature, and $[\text{ion}]_{\text{extracellular}}$ and $[\text{ion}]_{\text{intracellular}}$ are the ion concentrations outside and inside the cell, respectively.

The Nernst equation captures the dynamic balance between two competing forces: **the chemical gradient** and **the electrical gradient**.

1. **Chemical gradient**: This is the difference in ion concentration across a membrane. Ions tend to move from regions of high concentration to low concentration, driven by diffusion. If there's a higher concentration of a particular ion on one side of the membrane compared to the other, this gradient will push ions in the direction of lower concentration.

2. **Electrical gradient**: This is the potential difference across the membrane, created by the separation of charges. The membrane potential acts as an electrostatic force that either attracts or repels charged ions. For positively charged ions, the membrane potential will attract them toward the negative side and repel them from the positive side, and the opposite holds true for negatively charged ions.

At equilibrium, the forces reach a balance: the chemical force driving ions from high to low concentration is perfectly counteracted by the electrical force exerted by the membrane potential. The Nernst equation helps quantify the membrane potential at which this balance occurs, where the ion net movement stops because the two opposing forces are exactly equal and opposite. The resulting equilibrium potential for an ion reflects this conflict — if the membrane potential is less than the equilibrium potential, the electrical gradient will push positive (negative) ions into (out of) the cell, and if it's greater, positive (negative) ions will be pushed out of (into) the cell.

The Nernst potential as the motor of electrical activity Different ions have distinct Nernst potentials because of their unique concentration gradients and charges. Since the membrane permeability to each ion is dynamic, the overall membrane potential can shift depending on which ion channels are open. This creates excitability, as the membrane potential can move toward the Nernst potential of the dominant ion. Essentially, the Nernst potential acts as the voltage each ion 'wants' the membrane potential to reach, while the conductance of the ion channels regulates the intensity of the currents, driving ion movement and electrical activity.

2.1.2 Neuronal Degeneracy

Degeneracy is a fundamental property of biological systems, referring to the ability of structurally different elements to produce functionally equivalent outputs. This concept extends across multiple levels of organization in the nervous system, from ion channels to whole networks, providing robustness and adaptability in the face of genetic variability, environmental changes, and perturbations such as injury or disease.

Two concepts are often associated with **degeneracy** but are fundamentally distinct: **redundancy** and **insensitivity**. Redundancy refers to the existence of multiple copies of the same structure, allowing function to persist even if one copy fails. While redundancy provides resilience, it does not confer adaptability, as a global external perturbation affects all copies similarly. Insensitivity, on the other hand, characterizes situations where a system component does not contribute to the output. Degeneracy can sometimes be confused with insensitivity, as the output may appear unaffected by the system composition. However, in the case of degeneracy, compensatory mechanisms ensure that variations, when balanced, do not alter the output. In a neuron, *degeneracy occurs through ion channels* — we record neurons with similar electrical activity but with a widely different composition of ion channels, sometimes varying by several times (Marder, 2011).

It is difficult to discuss the concept of degeneracy without invoking the notion of models since the straightforward definition — "multiple sets of parameters yielding the same result" — directly refers to parameters, which are inherently tied to a particular representation of reality. In line with Goaillard and Marder, 2021, this section approaches degeneracy from a biological perspective rather than through the lens of a specific model. The aim is to emphasize that degeneracy is not a modeling artifact but a fundamental biological reality — one that should ideally be reflected in models.

Functional Overlap and Compensation

The concepts of functional overlap and compensation are central to the manifestation of degeneracy. Functional overlap means that different components — though structurally distinct — can perform similar roles in contributing to a system output. This allows one component to compensate when another is perturbed or impaired, preserving function. Functional overlap relates to an abstract level of output: the specific underlying mechanism is secondary to its effect on the biological system state. In this work, we focus on cellular-level degeneracy, specifically concerning membrane potential activity — we consider the *spike times* as the output of interest.

To illustrate the concept of degeneracy in a non-biological context, consider the example of an electric bicycle. An e-bike can move forward through pedaling, motor assistance, or a combination of both. These modes of propulsion represent **structurally distinct mechanisms**, yet they lead to the **same functional outcome**: forward motion. If the motor fails, the rider can continue by pedaling. Conversely, if the rider stops pedaling — due to fatigue, for example — the motor can compensate to maintain movement. This example captures the essence of **functional overlap** and **compensatory mechanisms**, which form the basis of degeneracy: different components that can substitute for one another to preserve function. Moreover, when observing the bike from a distance, it is impossible to determine the exact contribution of the motor or the pedaling to the overall movement. This lack of clarity regarding individual contributions, despite a stable overall output, mirrors how degeneracy can mask structural variability behind a preserved functional outcome.

In the context of neuron, several factors influence membrane potential, among which ion channels play a crucial role (Goaillard and Marder, 2021; Marder and Taylor, 2011). Although ion channels are typically specific to certain ions, the determinant of membrane potential is electric charge, irrespective of ion type. This explains the presence of functional overlap: the function in question is charge transfer with precise dynamics, governed by charge quantity and temporal properties.

Thus, as long as charge movement — regardless of ion type — is maintained, neuronal activity can persist. This highlights the existence of compensatory mechanisms, where a decrease in the effective quantity of one type of ion channel can be offset by an increase or decrease in another type.

However, while this view helps build intuition about degeneracy through ion channels, it does not capture the full picture. Although charge directly influences membrane potential, there are also indirect mechanisms and cascades of cellular reactions triggered by specific ions that can modulate electrical activity. This is particularly well-documented for calcium ions (Berridge, 1998).

Qualifying degeneracy

Degeneracy is inherently tied to neuronal function, making it essential to understand function before discussing degeneracy. For instance, two electrical activities that differ only slightly — such as a minor variation in spike frequency — might be considered functionally equivalent, implying degeneracy. This is particularly relevant when considering biological phenomena with inherent noise. This notion of "*sufficiently similar*" activity is introduced here as **degenerate proximity**. In other words, the degenerate proximity is the range within which variations in neuronal activity are considered negligible and thus the activities are functionally equivalent.

These definitions of degeneracy and degenerate proximity are fundamental to this work and are not universally adopted in the literature (Naudin, 2023). In parallel, we introduce the concept of **level of**

degeneracy, which refers to how easily degeneracy can be observed. Mathematically, this can be captured by the size of parameter space regions corresponding to activity with close degenerate proximity.

Degenerate proximity focuses on the "*external*" (or *observable*) aspects of the model, in the sense that it is related to the function or the trace of the neuron. It evaluates how similar the neuronal activity patterns are, regardless of the underlying parameters. On the other hand, the level of degeneracy pertains to the "*internal*" aspects of the model, as it is related to the parameters, such as maximal conductances. It assesses how extensive the regions in the parameter space are that produce functionally equivalent outputs.

Box II – Neurons in a Nutshell

Neurons are Excitable Cells

Neurons communicate by generating and propagating action potentials, which result from the controlled movement of ions across the membrane. This process is regulated by ion channels, specialized proteins that selectively allow ions to pass through, shaping the neuron electrical activity. The precise timing of these action potentials, known as spike times, carries information, as neural computation and signaling depend not only on whether a neuron fires but also on when it fires.

Neurons Exhibit Degeneracy

Degeneracy allows neurons to maintain similar spiking activity despite differences in their underlying ion channel compositions. This ensures robustness by enabling multiple parameter configurations to produce the same functional output. In this work, we focus on handling degeneracy by generating populations of neuron models that exhibit equivalent spiking patterns, ensuring that different sets of parameters lead to the same temporal structure of neuronal activity.

Ion Channels Define Firing Patterns

The firing patterns of neurons are determined by the specific properties, kinetics, and density of the ion channels they contain. Different ion channels contribute to the distinct phases of the action potential and the overall pattern of neuronal firing, shaping the information encoded by spiking activity. They are the main subject of interest in this work and are considered important variables for studying the electrical activity of a neuron.

2.2 Conductance-based Models

2.2.1 Generalities

Conductance-based models (CBMs) play a central role in this work. These models are capable of reproducing many different nonlinear neuronal behaviors, such as bursting, bistability, and others, while offering some biological interpretability. Their importance in computational neuroscience lies not only in their ability to capture the underlying mechanisms of neuronal excitability, but also in their scientific history.

In 1963, Sir John Carew Eccles, Alan Lloyd Hodgkin and Andrew Fielding Huxley were jointly awarded the Nobel Prize for their discoveries concerning the ionic mechanisms involved in excitation and inhibition in peripheral and central cell membranes (Nobel Prize Outreach, 1963). In their seminal work Hodgkin and Huxley, 1952, explored neuronal conduction mechanisms and the involvement of ionic currents in them, using the giant squid axon as an experimental model. This work laid the foundations for modern CBMs, which continue to evolve while retaining a similar fundamental structure.

Basic CBM structure At their heart, CBMs transcribe the electrical dynamics of the neuronal membrane into a set of coupled ordinary differential equations (ODEs). Conceptually, each membrane surface element is assimilated to an equivalent *nonlinear* RC electrical circuit, the schematic of which is shown in Figure 2.5, composed mainly of two elements:

- A capacitance (C): Representing the membrane phospholipid bilayer, almost impermeable to ions, along with the leakage current $I_{\text{leak}} = g_{\text{leak}}(V - E_{\text{leak}})$, which encodes the small membrane leakage.
- Ion channel conductances ($g_i(V,t)$): Modeling ion channels and other membrane proteins that allow the selective passage of ions across the membrane. The complexity of neurons arises, in part, due to the presence of *voltage-gated* channels, which are sensitive to changes in membrane potential and significantly influence neuronal behavior.

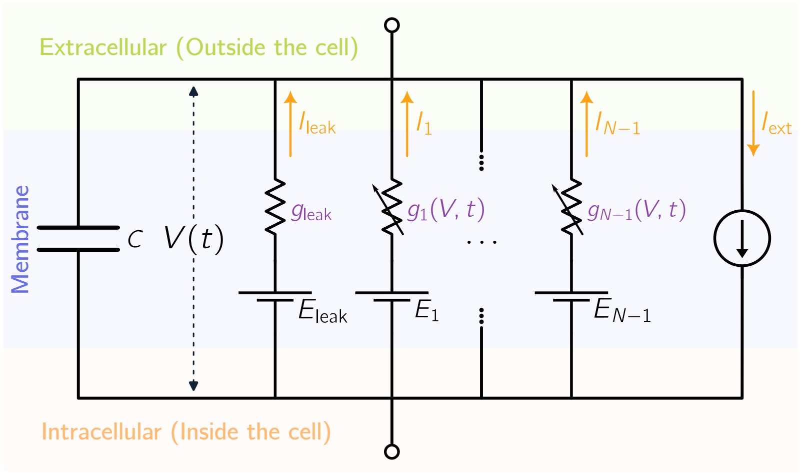


Figure 2.5: Equivalent electrical circuit of Conductance-Based Models (CBMs). The membrane capacitance C represents the phospholipid bilayer, which is impermeable to ions. The circuit includes $N-1$ voltage-dependent ionic conductances $g_i(V,t)$, each associated with a reversal potential E_i, as well as a leakage conductance g_{leak} with its corresponding reversal potential E_{leak}. The total membrane current consists of the ionic currents I_i flowing through their respective conductances and the passive leakage current I_{leak}. An external current I_{ext} represents input stimuli or injected current.

The conductances in CBMs are not constant; they are variable and depend on both the membrane potential and time. The membrane voltage defines a steady-state value toward which the conductance evolves, and the time dependency accounts for the dynamics of this evolution toward the steady-state. Intuitively, the changing conductance values model the opening and closing of ion channels. These conductances are nonlinear, meaning their relationship with voltage is not proportional, which is crucial for accurately capturing excitability.

The application of fundamental electrical circuit laws (notably Kirchhoff's voltage and current laws) allows us to derive the equations governing the dynamics of currents and membrane potential. For example, in its simplest form, the temporal variation of membrane potential V, can be expressed as follows:

$$C\dot{V} + g_{\text{leak}}(V - E_{\text{leak}}) = -\sum_{i=1}^{N_{\text{model}}-1} g_i(V,t)(V - E_i) + I_{\text{ext}}, \tag{2.2}$$

where:

- C is the membrane capacitance per unit of membrane area, usually expressed in $\mu\mathrm{F\,cm^{-2}}$. Very often, the value $C = 1\,\mu\mathrm{F\,cm^{-2}}$ is used[2].

- $g_i(V,t)$ represents the voltage-dependent ionic conductances, expressed in $\mathrm{mS\,cm^{-2}}$. Particularly, g_{leak} is the passive leakage constant conductance.

- E_i denotes the reversal potentials — or Nernst potentials — associated with each ion, expressed in mV.

- I_{ext} corresponds to the externally injected currents, expressed in $\mu\mathrm{A\,cm^{-2}}$. In this work, we consider the spontaneous activities of neurons, and thus we set $I_{\text{ext}} = 0$.

General formulation of conductances The conductance $g_i(V,t)$ associated with a specific ion channel can be expressed as follows:

$$g_i = \bar{g}_i m_i^{p_i} h_i^{q_i}, \tag{2.3}$$

where:

- $\bar{g}_i \in \mathbb{R}_+$ is the maximal conductance of the ion channel. This usually constant parameter represents the **maximum** effective permeability of the membrane to specific ions when all corresponding channels are open.

- m_i and $h_i \in [0,1]$, are activation and inactivation gating variables, respectively, describing the state of ion channels particles (open or closed).

- p_i and $q_i \in \mathbb{N}$ are constant integer exponents that reflect the number of particles required to fully activate or inactivate the channel.

The product $m_i^{p_i}(V,t)h_i^{q_i}(V,t)$ thus models the probability that the channels are open at a given moment. Consequently, the value of $g_i(V,t)$ varies based on these probabilities, allowing for the temporal dynamics of ionic currents to be captured. The fact that we have a lot of conductance leads to a lot of laws. So the high dimensionality of the parameters also increases what can be represented by the model.

In practice, the parameters p_i and q_i are often determined through a *fitting* process using experimental data, but they can also sometimes be related to biology. Indeed, ion channels are transmembrane proteins composed of multiple, often symmetric, subunits (Hammond, 2024, Chapters 3–5). Each subunit contains one or more pore regions that form the pathway across the membrane. These subunits can be activated or inactivated through structural changes in recognition domains, which are often located in extra- or intra-membrane regions sensitive to the membrane potential. The activation of an ion channel often requires multiple activation sites to be occupied simultaneously. Similarly, inactivation may involve multiple inactivation sites. These biological mechanisms are directly reflected in the values of p_i and q_i.

[2]An alternative is to express all conductances in $\mathrm{mS\,\mu F^{-1}}$, which is common in computational neuroscience research papers.

In the context of CBMs, it is common to use the term "*ionic current*" to refer to the product : $I_i = g_i(V,t)(V - E_i)$. By convention, a positive ionic current corresponds to an outward flow of positive charge (or an inward flow of negative charge), while a negative current corresponds to an inward flow of positive charge (or an outward flow of negative charge). Thus, the classical structure of CBMs, as described in Figure 2.5, leads to a dynamic of the membrane potential — also sometimes called the capacitive current $I_{\text{capacitive}} = C\dot{V}$ — which is given by the sum of ionic and external currents, injected into the circuit, as described by the Equation (2.2).

Dynamics of activation and inactivation variables The variables m_i and h_i follow first-order differential equations that describe their temporal evolution as a function of membrane potential V. This is why we sometimes explicitly write a time and membrane potential dependence for the conductances $g_i(V,t)$. These equations generally take the following form:

$$\tau_{X_i}(V)\dot{X}_i = X_{i,\infty}(V) - X_i, \tag{2.4}$$

where X_i represents either m_i or h_i, and :

- $\tau_{X_i}(V)$ is the voltage-dependent time constant that governs how quickly X_i approaches its steady-state value.

- $X_{i,\infty}(V)$ is the voltage-dependent steady-state value of X_i, typically described by a sigmoidal function that reflects the probability of ion channel opening or closing.

This formulation captures two essential aspects:

1. **The fast or slow dynamic of channels**: The time constant $\tau_{X_i}(V)$ can vary significantly depending on the type of channel and the membrane potential. For example, sodium channels activate very quickly during depolarization, while potassium channels have a slower kinetics.

   In the rest of this work, the timescale separation of channels will be at the core of the DICs theory, presented in Section 2.3. Without such a difference in channel timescales, provided by biology, neuronal behaviors would probably be much less rich (Franci et al., 2014; Hille, 1984). Thus, although the functions $\tau_{X_i}(V)$ do not influence the equilibrium points of the system described by the CBM — that is, the set of values of V and X_i for which all temporal derivatives vanish — their importance should not be overlooked.

2. **The nonlinear dependence on V of activation and inactivation variables**: The function $X_{i,\infty}(V)$ generally follows a sigmoidal curve, representing the probability of channel opening as a function of membrane potential. This nonlinear relationship is essential for finely modulating neuronal excitability and giving rise to complex dynamic behaviors such as adaptation, resonance, or neuronal oscillations (Izhikevich, 2007).

While the standard formulation assumes that conductances depend primarily on voltage, some models incorporate additional dependencies on ion concentrations. In such cases, the steady-state gating functions $X_{i,\infty}$ may depend on ion concentrations in addition to voltage. Ion concentrations can be explicitly modeled as dynamic variables, or they may only influence gating functions without being directly represented in the model. For instance, in the stomatogastric ganglion (STG) neuron model considered in this work (Section 2.2.2), calcium ion plays an active role, with both Ca-dependent gating functions and explicit Ca dynamic included in the model. In contrast, in the dopaminergic (DA) neuron model (Section 2.2.3), magnesium ion influences gating variables, yet Mg concentration is not explicitly modeled as a dynamic variable.

A passive and non-specific conductance: the leak conductance In addition to the specific conductances associated with activated or inactivated ion channels, CBMs often include a passive conductance, commonly referred to as the leak conductance g_{leak}. The passive nature of this conductance lies in the fact that it is voltage-independent and ion-independent. Thus, although we use the general notation $g_i(V, t)$, in the case of g_{leak}, we have $g_{\text{leak}}(V, t) = g_{\text{leak}}$.

Biologically, this conductance can be considered *non-specific* to a given ion in the sense that it corresponds to a non-selective passage of ions across the membrane. This conductance is often included in CBMs to regulate the membrane potential and to model the aggregation of numerous ions that are not explicitly represented. The leak conductance is associated with a Nernst potential E_{leak}, which accounts for the different ions. An alternative interpretation is that leakage conductance models the neuron input resistance when at rest.

Importance of the maximal conductance parameters The parameters $\bar{g}_i$, often referred to as *maximal conductances*, are among the most critical elements in CBMs. These values represent the maximum permeability of the membrane to specific ions when all corresponding channels are open. Their role extends far beyond simple biophysical characterization: they directly influence the dynamic behavior of neurons and, by extension, the functional properties of neural networks.

In CBMs, the $\bar{g}_i$ are considered fundamental parameters of the model, as they control the intensity of the ionic currents crossing the neuronal membrane and, consequently, regulate excitability processes and the integration of synaptic signals. Their precise tuning is essential for accurately reproducing the dynamics observed in biological neurons, whether in response to stimuli, oscillatory behavior, or participation in neuronal communication networks.

Although the Nernst potentials of the involved ions are important in modeling, they are generally determined by intra- and extracellular ion concentrations and do not vary significantly in biological systems, whereas the $\bar{g}_i$ is more variable and subject to optimization.

Thus, the $\bar{g}_i$ are the parameters to be determined by experimentalists, as discussed in the introductory Section 1.1 of this work. These $\bar{g}_i$ values can lead to degeneracy and are particularly difficult to determine in order to reproduce an observed behavior. Their distributions, conditioned on neuronal activity, constitute the ultimate objective of this work. What makes them so complex to identify is the intractable nature of CBMs, i.e. without simulating the model, we cannot know *a priori* the corresponding activity.

Finally, it is common in CBMs to distinguish the $\bar{g}_i$ from g_{leak} due to their biological differences and their respective roles in the CBM. However, in the remainder of this work, we will use $\bar{g}_i$ to refer to both the maximal conductances of specific ion channels and $g_{\text{leak}} \triangleq \bar{g}_{\text{leak}}$, unless otherwise specified. Thus, when referring to a specific instance of a CBM associated with its parameter vector, we will use $\bar{\boldsymbol{g}} = \left[\bar{g}_1, \bar{g}_2, \ldots, \bar{g}_{N_{\text{model}}-1}, g_{\text{leak}}\right] \in \mathbb{R}_+^{N_{\text{model}}}$.

About the biological interpretability of the maximal conductances A common misconception in CBMs is the direct interpretation of $\bar{g}_i$, the maximal conductance, as the density of ion channels on the membrane. While channel density and $\bar{g}_i$ are not unrelated, this interpretation is incomplete. Rather than reflecting a strict count of ion channels, $\bar{g}_i$ should be understood as an *effective* density, integrating various regulatory mechanisms that modulate channel activity beyond their exact number. Processes such as phosphorylation, intracellular signaling, and channel trafficking dynamically influence ion channel function, altering their open probability, gating kinetics, or localization within the membrane. These modulations can lead to significant variations in $\bar{g}_i$ without changes in the total number of channels. Thus, in the context of CBMs, $\bar{g}_i$ encapsulates the overall functional availability of ion channels rather than merely their physical distribution. Since the biological foundations of these mechanisms have already been briefly discussed in Section 2.1, we emphasize here their modeling implications: $\bar{g}_i$ is not a fixed anatomical property but rather a parameter that integrates multiple layers of cellular regulation.

About varying maximal conductances In the remainder of this work, we consider the maximal conductances as fixed parameters of the model. However, a fundamental aspect of neuronal function lies in their ability to be modulated over time, whether through homeostatic plasticity mechanisms, activity-dependent regulation, or metabolic influences.

For example, Liu et al., 1998 proposed a model with calcium dynamics that can induce a slow adaptation of maximal conductances, providing a feedback mechanism that stabilizes neuronal excitability over long periods. Similarly, O'Leary et al., 2014 studied conductance-based models within a homeostatic regulation framework, illustrating how neurons can maintain functional stability despite parameter variability by adjusting the values of $\bar{g}_i$.

A particularly interesting recent example is that of Fyon et al., 2023, which proposes an adaptive neuromodulation model based on the theory of DICs — the same theoretical framework used in this work that will be presented in Section 2.3. This study demonstrates how an adaptive control of ion channels, through a simple feedback loop, allows the regulation of neuronal excitability despite the degeneracy of maximal conductances. Furthermore, the proposed method is compatible with the homeostatic control model of O'Leary et al., 2014, as shown in a subsequent study by Fyon and Drion, 2024.

These results suggest that a neuron conductance profile is not fixed but rather shaped by its activity history and intracellular signaling pathways.

While the fundamental principles of CBMs presented above apply universally, their implementation varies depending on the biological systems under study. To illustrate the generality of our method, we will apply our approach to two distinct models: a model of the stomatogastric ganglion (STG) neurons and a model of dopaminergic (DA) neurons. These two models are presented in detail in the following.

Finally, readers less familiar with CBMs may find Box III useful, as it provides a concise summary of the key equations and components discussed in this section.

Box III – How to Properly Define a Conductance-Based Model?

The Model Structure

Conductance-based models (CBMs) provide biophysically realistic representations of neuronal activity using equivalent circuit theory (Fig. 2.5). A CBM requires defining:

- **Ionic Currents:** A set $\{I_i\}_{i=1}^{N_{\text{model}}-1}$, each expressed as:

  $$I_i = \bar{g}_i m_i^{p_i} h_i^{q_i} (V - E_i),$$

  where:
  - $\bar{g}_i$: Maximal conductance.
  - E_i: Nernst potential.
  - m_i, h_i: Gating variables.
  - p_i, q_i: Gating exponents.
- **Gating Dynamics:** For each gating variable X_i:
  - $X_{i,\infty}(V)$: Steady-state gating function.
  - $\tau_{X_i}(V)$: Time constant governing X_i.
- **Membrane Potential Dynamics:** Given by:

  $$C\dot{V} + I_{\text{leak}} = -\sum_{i=1}^{N_{\text{model}}-1} I_i + I_{\text{ext}},$$

  where C is membrane capacitance, I_{ext} is external input and $I_{\text{leak}} = g_{\text{leak}}(V - E_{\text{leak}})$ is the leak current.
- **Optional Extensions:** Additional ODEs for explicit ion dynamics or other state variables.

The complete CBM is described by:

$$C\dot{V} = -\sum_{i=1}^{N_{\text{model}}} \bar{g}_i m_i^{p_i} h_i^{q_i} (V - E_i) + I_{\text{ext}},$$

$$\tau_i(V)\dot{X}_i = X_{i,\infty}(V) - X_i.$$

This framework models essential neuronal properties and dynamic behaviors.

An Instance of the Model

In this work, we assume that the only difference between two instances of the same model lies in their parameter vector:

$$\bar{\boldsymbol{g}}_{\text{model}} = \left[\bar{g}_1, \bar{g}_2, \ldots, \bar{g}_{N_{\text{model}}-1}, g_{\text{leak}}\right] \in \mathbb{R}_+^{N_{\text{model}}}$$

Thus, an instance is fully defined by the model structure and $\bar{\boldsymbol{g}}_{\text{model}}$ (or simply $\bar{\boldsymbol{g}}$ when unambiguous).

2.2.2 A Stomatogastric Ganglion Neuron Model

The stomatogastric ganglion (STG) is a well-studied neural structure found in crustaceans, responsible for generating rhythmic motor patterns that control digestive movements such as chewing and filtering (Bucher et al., 2006). Despite its relatively small size, the STG exhibits a rich repertoire of neuronal dynamics, making it a powerful model system for understanding fundamental principles of neuronal excitability, network oscillations, and neuromodulation. One of the major features of STG neurons is their ability to maintain stable spontaneous rhythmic activity despite significant variability in their underlying conductances — in other words, they are a good choice for studying the degeneracy phenomenon (Marder and Taylor, 2011).

The spontaneous activities, i.e. activities that occurs even if there is no input current or stimulus, observed vary between different neuron models. In the case of the STG model, as will be demonstrated later in this section, it can exhibit spontaneous *tonic spiking*, *bursting*, *one-spike bursting*, as well as slightly or highly *irregular* spontaneous activity. In comparison, the DA model, introduced in the following Section 2.2.3, is capable of generating *slow-pacemaking*, *bursting*, and *fast-spiking* behaviors.

The STG neuron model used in this study is adapted from Liu et al., 1998. The remainder of this section outlines its core components. Following Box III, the currents considered in the STG model include seven voltage-dependent currents — one of which is also calcium-dependent — along with a leakage current. A brief biological description and an overview of the dynamics of these currents are provided. The exact expressions for the *steady-state gating* functions, time constants, various exponents, and Nernst potentials can be found in Appendix B.1. Visual representations of these functions are available in Figures B.1 and B.2.

Sodium current I_{Na}; is characterized by very fast activation and inactivation. It is the primary driver of depolarization during an action potential. Increasing this current increases the model excitability to a certain extent.

Potassium currents I_{Kd}, I_{KCa} and I_{A}; Although they involve the same ion, they have distinct temporal properties and roles:

- I_{Kd} (Delayed Rectifier): Primarily responsible for repolarizing the neuron after depolarization. Its activation is considered slow compared to the dynamics of the sodium current.

- I_{KCa} (Calcium-Dependent Potassium Current): Modulated by intracellular calcium concentration, it plays a role in terminating bursts of activity and regulating their frequency.

  Special attention must be given to the explicit dependence of this current on intracellular calcium, expressed as $m_{\mathrm{KCa},\infty} = m_{\mathrm{KCa},\infty}(V, \mathrm{Ca})$. This dependence is directly incorporated into the STG model, where higher Ca concentrations promote stronger activation of I_{KCa}.

- I_{A} (Transient Potassium Current): Contributes to action potential frequency adaptation and delays their initiation.

Calcium currents I_{CaS} and I_{CaT}; These currents are associated with calcium ions, which are now recognized as playing a role just as significant as Na^{+} or K^{+}, although this was not always acknowledged historically.

- I_{CaS} (Slow Calcium Current): Facilitates a prolonged influx of Ca^{2+} and contributes to long-term activity regulation. It has relatively slow dynamics.

- I_{CaT} (Transient Calcium Current): Activates at lower potentials and plays a role in burst initiation. Its dynamics are faster than those of I_{CaS}.

Hyperpolarization-activated inward cation current I_H; This current is unique in several ways. First, it is associated with multiple cations and is therefore less specific. The involved cations are considered to be Na^+ and K^+, meaning its Nernst potential lies between the sodium and potassium potentials.

Additionally, this current is known as a *hyperpolarization-activated* inward cation current because its channels open at membrane potentials near or slightly more hyperpolarized than the resting potential. It inactivates at higher potential values. As shown in Figure B.1 (Appendix B.1), it is the only current in the model whose steady-state activation value, $m_{H,\infty}(V)$, decreases with membrane potential — a behavior typically observed in inactivation variables.

It plays a crucial role in regulating the resting potential and contributes to bursting activity. Its dynamic is slow compared to sodium and potassium currents.

Leak current I_{leak}; A passive leak current is included in the model.

Explicit calcium dynamics That is, Ca is treated as a state variable in the model, representing the <u>intracellular</u> concentration of Ca^{2+} (understood here as the *cytoplasmic* concentration). The extracellular concentration is assumed to remain constant. The differential equation (2.5) is added to the STG neuron CBM and reflects the fact that calcium currents directly influence the influx of Ca^{2+} into the cell[3], while various cellular mechanisms sequester and remove calcium from the cytoplasm:

$$\tau_{Ca}\dot{Ca} = -\alpha_{Ca}\left(I_{CaS} + I_{CaT}\right) - Ca + \beta_{Ca}, \tag{2.5}$$

where:

- τ_{Ca}, the time constant of calcium dynamics. Here, $\tau_{Ca} = 20\,\text{ms}$, making it a slow variable in the system.
- $\alpha_{Ca} = 0.94\,\text{mM}\,\text{nF}\,\text{nA}^{-1}$, which can be interpreted as the conversion factor between a calcium current and the change in intracellular calcium concentration.
- $\beta_{Ca} = 0.05\,\mu\text{M}$, the resting calcium concentration.

Thus, in line with the CBM framework, the calcium dynamics in the STG model remain highly interpretable from a biological perspective. Finally, readers are encouraged to read Appendix B.1, which details the implementation, including the differences with Liu et al., 1998 original work.

Observed Firing Patterns

Different types of activities can be observed in the STG model. Notably, Figures 2.6A and 2.6B illustrate instances of *tonic spiking* and *tonic bursting*, respectively — where the term *tonic* is often omitted in the remainder of this work. These two activities are of particular interest due to the fundamental difference in the structure of their *spike times* sequences.

Figure 2.7A illustrates an instance of *one-spike bursting*, a paradoxical term commonly used in the STG community to describe a single discharge followed by a silent period, during which the neuron state remains depolarized but does not reach full excitation.

Finally, Figures 2.7B and 2.7C depict behaviors that are slightly irregular and highly irregular, respectively. Slightly irregular behaviors are classified as *bursters*, while highly irregular behaviors are not examined in detail in this work. According to Prinz et al., 2003, the latter account for only 0.5% of their 1.7 million

[3]Following the sign conventions for ionic currents, since Ca^{2+} carries a positive charge, $I_{Ca} = I_{CaS} + I_{CaT} < 0$ corresponds to an inward movement of Ca^{2+} from the extracellular to the intracellular space.

observed behaviors. It may also be pointed out that these distinctions between types of activity allow for simpler description and discussion, but are by no means essential to this work, which is why no formal, quantitative definition has been introduced to distinguish between strongly and slightly irregular activities.

When examining Figures 2.6, 2.7, and 2.8, one can better visualize the challenges presented in the introduction: it is very complex to determine *a priori* the type of activity that will be observed based solely on the conductance values without simulating the model (**intractability of CBMs**); and there is no simple method to retrieve the possible conductance values from the trace or spike times alone (**linking records to parameter values**). This work focuses on the second challenge.

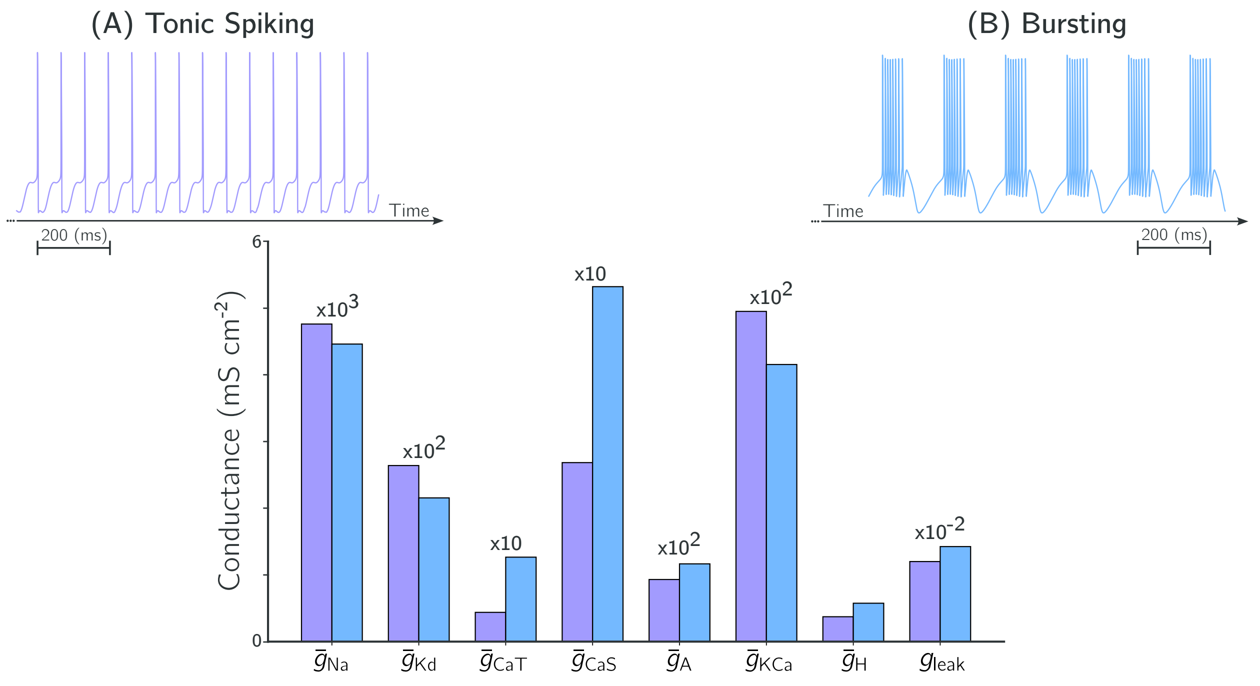


Figure 2.6: Tonic Spiking and Bursting in the STG Model. Tonic spiking (purple) and bursting (blue) are typical voltage traces of the STG model shown in the top panels (A) and (B), along with their associated conductance values displayed in the bottom panels. Each bar should be scaled relative to its indicated factor, which is shown above it. For instance, the sodium conductance for spiking is approximately 5000 mS cm^{-2}.

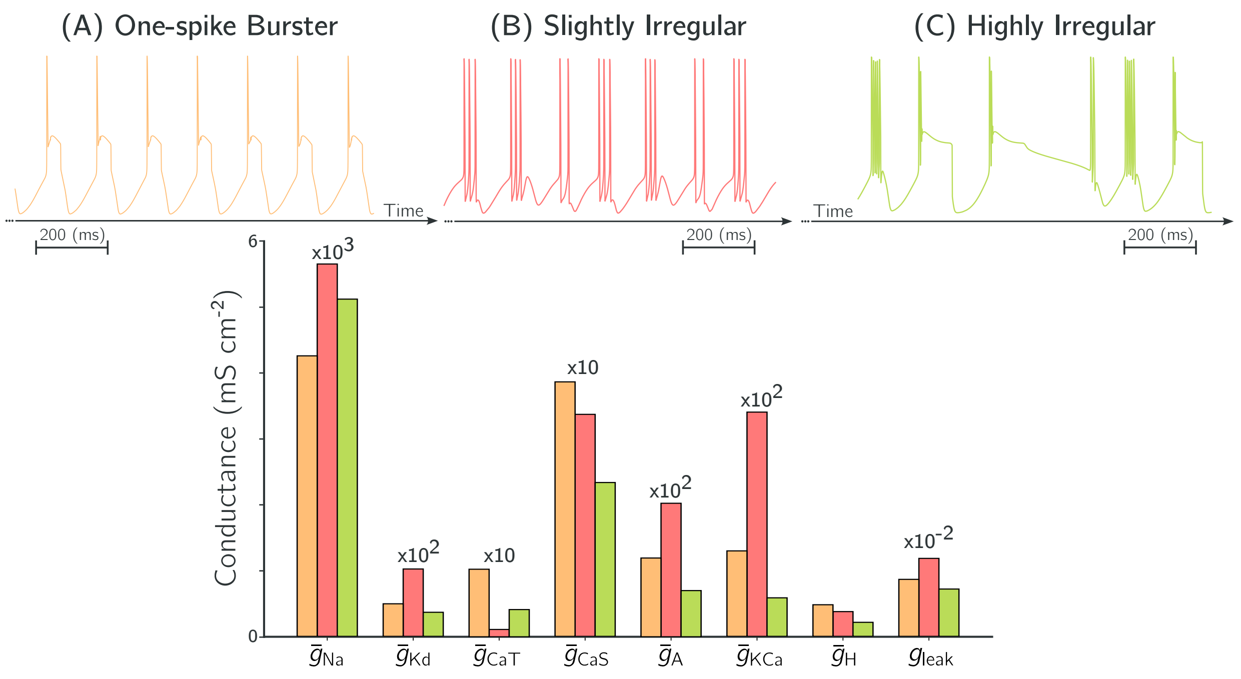


Figure 2.7: Irregular and One-Spike Bursting Activity in the STG Model. The top panels illustrate different neuronal firing behaviors: (A) One-spike bursting (orange), (B) Slightly irregular activity (red), and (C) Highly irregular activity (green). The bottom panel presents the corresponding conductance values for each case. Each bar should be interpreted relative to its indicated scaling factor as in Fig. 2.6. It is very difficult to predict the observed behavior solely based on conductance values, highlighting the intractable nature of CBMs.

2.2.3 A Dopaminergic Neuron Model

The second neuron model used in this work is a midbrain dopaminergic (DA) neuron model. These neurons are known for their highly stable and slow firing rate and are thus classified as *pacemaker neurons*. Their primary function is the release of dopamine. We use a slight variant of the model proposed by Qian et al., 2014[4].

The entire model is described by seven conductances, including the leak conductance. These conductances are g_{Na}, g_{Kd}, g_{CaL}, g_{CaN}, g_{ERG}, g_{NMDA}, and g_{leak}. It is worth noting that there is an explicit dependence on the extracellular magnesium concentration in the equation of the NMDA conductance. However, unlike calcium in the STG model, magnesium is considered constant here, and thus no dynamic is introduced. Interested readers are referred to Appendix B.2 for a more detailed presentation of the model and its equations.

Observed Firing Patterns

The DA model used exhibits three major types of activity: (A) *slow (spiking) pacemaking*, which involves relatively slow spiking activity compared to the spiking observed in the STG model (around 1 Hz for the DA model compared to 15 Hz for the STG model); (B) *bursting*, which can present a much larger number of spikes per burst compared to the STG model; and (C) *fast spiking*, characterized by spiking at a very high frequency. In practice, this type of activity is pathological and can be considered as a degenerate (non-ending) bursting; however, since the model can represent it, we retain it. Figure 2.8 illustrates each of these activities with a typical trace.

[4]The variant involves removing the SK channels from the model, following Fyon et al., 2024.

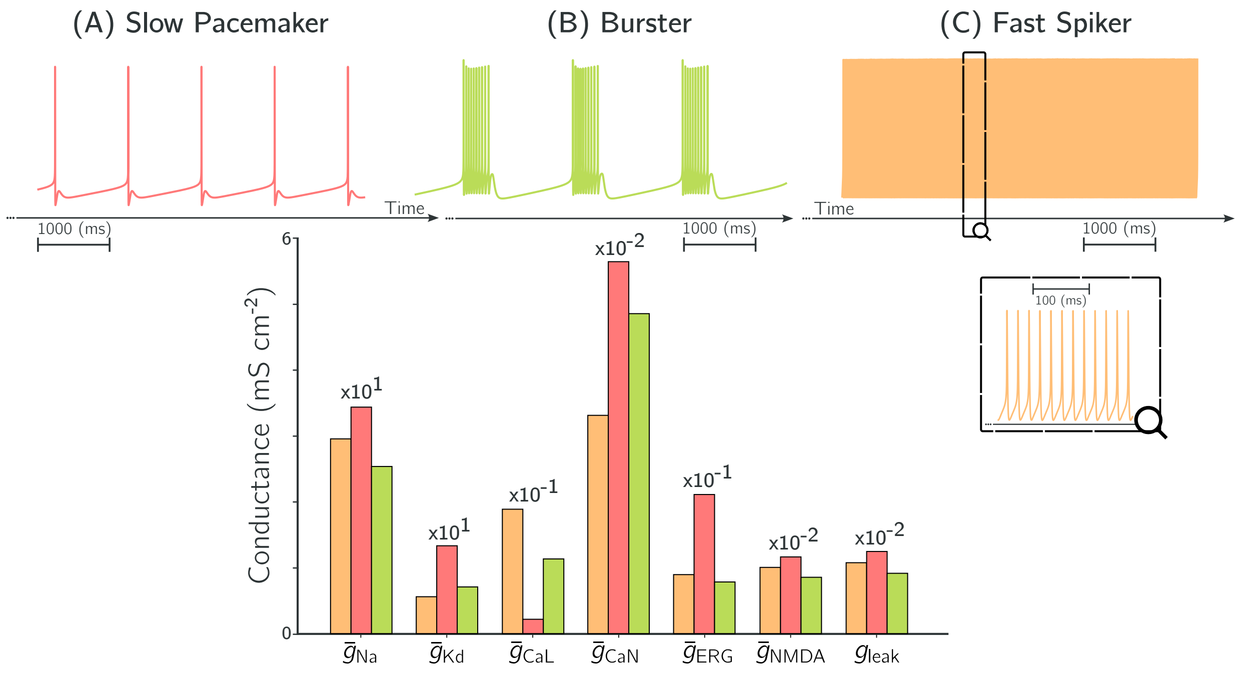


Figure 2.8: Different Activity Patterns in the DA Neuron Model. The figure illustrates various neuronal firing behaviors in the DA neuron model: (A) Slow pacemaking (red), showing relatively slow and consistent spiking activity; (B) Bursting (green), featuring clusters of spikes; and (C) Fast spiking (orange), characterized by high-frequency spiking. The bottom panel presents the corresponding conductance values for each case. Each bar should be interpreted relative to its indicated scaling factor as in Fig. 2.6.

2.3 The Theory of Dynamic Input Conductances

As explained in the introduction, estimating the parameters of conductance-based models is a complex problem due to the high dimensionality of parameter spaces and their degeneracy, combined with the intractable nature of CBMs. This complexity makes it difficult to precisely identify optimal parameter configurations and necessitates the development of new analytical approaches (Marder and Taylor, 2011).

In this context, the theory of *Dynamic Input Conductances* (DICs), introduced by Drion et al., 2015, provides a mathematical framework for analyzing the effects of parameter variations on CBM dynamics. This approach extracts essential information about neuronal excitability by characterizing how ionic conductances contribute to spiking and bursting properties. It is a way of overcoming the intractability of CBMs.

The concept behind DICs The fundamental idea behind DICs is to decompose membrane dynamics into different timescales. This decomposition establishes a correspondence between ionic currents and their effects at different temporal levels of neuronal activity. Instead of analyzing each biophysical parameter individually, their effects are grouped into voltage-dependent conductance curves. In the case of bursting activity, three distinct timescales are required: **fast** $g_{\mathrm{f}}(V)$, **slow** $g_{\mathrm{s}}(V)$, and **ultra-slow** $g_{\mathrm{u}}(V)$.

In static conditions, the membrane potential is at equilibrium — meaning no capacitive current is present — and Kirchhoff's laws reduce to an algebraic system. This contrasts with the dynamic nature of the membrane potential during an action potential, which leads to differential relations such as equation (2.2). The static approach has the major advantage of allowing direct sensitivity analysis. Indeed, Ohm's law then applies, and conductance — referred to as *static* in this case — determines the voltage-current relationship:

$$\Delta V \approx \frac{\partial V}{\partial I}\Delta I, \quad g_{\mathrm{static}}(V) = -\left(\frac{\partial I}{\partial V}\right)^{-1} \tag{2.6}$$

The DICs theory extends this idea by considering dynamic conditions, relying on the separation of membrane dynamics into different timescales. This separation allows for approximate relations — termed *dynamic input conductances* — that describe the voltage-current relationship in an algebraic manner. While it is conceptually possible to consider any number of timescales, practical applications typically require only a few. For bursting behavior, specifically, three timescales are sufficient. This means that the description of any CBM with N_{model} conductances can be reduced to just three voltage-dependent curves through DICs theory. Figure 2.9 illustrates this principle, showing how a complex CBM is reduced to a simplified model, which is interpretable, with only three dynamic conductances — the details of this reduction are explained later in this section. Furthermore, Drion et al., 2015 observed that a complete description of these three curves is not always necessary; a large portion of neuronal activity can be explained using only their values around a particular membrane potential, known as the *threshold potential* V_{th}:

$$\underbrace{\bar{\boldsymbol{g}} \in \mathbb{R}_+^{N_{\mathrm{model}}} \xrightarrow{\text{Dynamic Input Conductances}} \boldsymbol{g}_{\mathrm{DICs}}(V) = \begin{bmatrix} g_{\mathrm{f}}(V) \\ g_{\mathrm{s}}(V) \\ g_{\mathrm{u}}(V) \end{bmatrix} : \mathbb{R} \to \mathbb{R}^3 \xrightarrow{\text{Threshold Potential } V_{\mathrm{th}}} \boldsymbol{g}_{\mathrm{DICs}}(V_{\mathrm{th}}) \in \mathbb{R}^3}_{\texttt{Reducing the dimensionality of CBMs through DICs, from } \mathbb{R}_+^{N_{\mathrm{model}}} \texttt{ to } \mathbb{R}^3.} \tag{2.7}$$

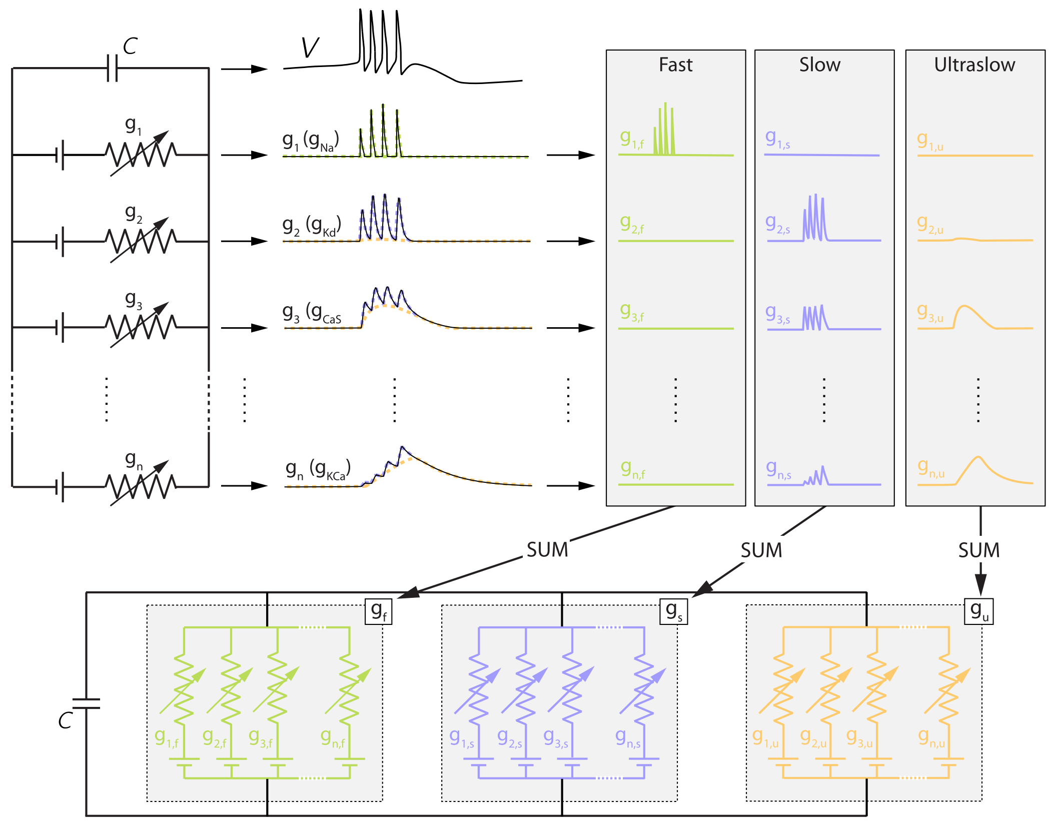


Figure 2.9: Reducing the dimensionality of an arbitrary CBM through the theory of DICs. Illustration of a conductance-based model and its dimensionality reduction via dynamic input conductances (DICs). The original high-dimensional model (top left) generates membrane potential and conductance traces over time (top center). These signals are decomposed into three distinct timescales — fast, slow, and ultra-slow (top right). The bottom panel reconstructs the model by grouping ion channel contributions by timescale into the dynamic input conductances g_f, g_s and g_u. The resulting model is a reduced, interpretable version of the original — *Adapted from*[a].

[a]Drion et al., 2015, Figure 2

Neurons as mixed-feedback systems DICs theory introduces an alternative way to approach neuronal behavior. Instead of describing neurons in terms of numerous individual currents, the system can be viewed as an interplay of feedback mechanisms. This perspective is at the core of many recent studies in computational neuroscience and neuromorphic engineering. Two principles emerge from mixed-feedback systems, as highlighted by Sepulchre et al., 2019:

- Feedback shapes the sensitivity of an input-output behavior. Positive feedback is a source of ultrasensitvity, memory, and discrete signaling. Negative feedback is a source of infrasensitivity, linearity, and continuous signaling.

- Excitability is a mixed-feedback mechanism.

Through DICs theory, CBMs can be simplified into a set of three feedback gain curves[5] (fast, slow, and ultra-slow), which result from the aggregation of different system components (ionic currents). This approach highlights that the exact composition of these components is less important than the final feedback curves themselves. The insight is that neuronal behavior is shaped by the net effect of different

[5]With the sign convention used (following Fyon et al., 2024), the DIC curves must first be changed of sign to obtain the true sign of the feedback. Gain is the absolute value of the DIC. For example, a negative value of $g_f(V)$ indicates fast positive feedback at the potential value in question.

feedback interactions rather than by individual conductances. This idea is fundamental to the compensation algorithm introduced later in this section.

Applying the DICs theory Figure 2.10 illustrates the DIC curves obtained for two instances of the STG model: one in spiking mode and one in bursting mode. These curves can be analyzed in the same way as conventional *I*-*V* relationships, but across different timescales. Note that, in practice, the timescale influences the relevant portion of the *I*-*V* relationship. For instance, while $g_\mathrm{u}(V)$ exhibits highly negative conductance — indicating positive feedback — at high voltages, these voltage excursions occur on much faster timescales than the ultra-slow component. As a result, g_u is not effectively recruited in this range. Readers are referred to Appendix C for further details on DICs.

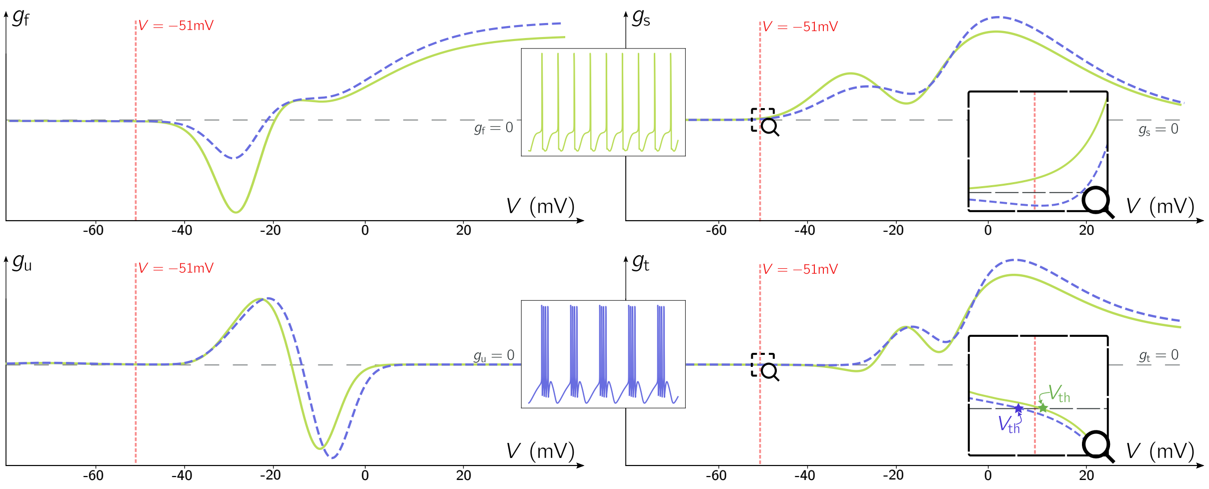


Figure 2.10: Dynamic Input Conductances (DICs) curves for two instances of the STG model: one spiking and one bursting. The dashed purple line represents bursting, while the solid green line represents spiking. The red vertical dashed line indicates $V = -51\,\mathrm{mV}$, a reference value used throughout this work. The top left panel shows g_f, which is mostly negative for negative potentials and becomes positive as V increases, reflecting the passive properties of the membrane. The top right panel presents g_s, the fundamental difference between spiking and bursting. While generally positive, a closer look around $V = -51\,\mathrm{mV}$ reveals a small negative region for bursting, allowing sufficient positive feedback for bursting to occur, whereas in spiking it remains positive. The bottom left panel displays g_u, which is always positive; the negative portion at higher potentials is not relevant, as the ultra-slow timescale is not effectively engaged in this range. The bottom right panel shows g_t, the total conductance, where the threshold potential (marked by stars) indicates the transition from a net negative feedback to a net positive feedback. This threshold varies between instances but is generally distributed around $V = -51\,\mathrm{mV}$. The values of the DICs around V_th play a crucial role in shaping neuronal activity.

In summary, DIC curves provide a concrete representation of the feedback-based description of spiking and bursting, as developed in Franci et al., 2013, 2014, 2018. Specifically, we can qualitatively analyze Figure 2.10 to understand the fundamental differences between those activity types:

1. Both spiking and bursting exhibit predominantly positive fast feedback, which drives spontaneous depolarization in both cases. This is evident in Figure 2.10 (top left), where $g_\mathrm{f}(V)$ is negative at subthreshold voltages, reinforcing action potential initiation. At higher voltages, $g_\mathrm{f}(V)$ becomes positive, reflecting the negative feedback from the passive properties of the membrane, which contributes to action potential termination.

2. The key distinction between spiking and bursting arises in the slow feedback component. In the spiking regime, $g_\mathrm{s}(V)$ remains positive throughout, ensuring a stabilizing effect that limits repetitive firing to individual action potentials. In contrast, in the bursting regime, there is a localized region

where $g_s(V)$ becomes slightly negative (Fig. 2.10, top right), enabling a transient positive feedback loop that sustains burst activity before ultra-slow repolarization terminates the burst.

3. The ultra-slow feedback component, $g_u(V)$, plays a critical role in burst adaptation. Although it remains positive for most voltage values (Fig. 2.10, bottom left), its relative strength influences burst duration and interburst intervals. The negative feedback of the ultra-slow timescale ensures a burst termination.

Mathematical Description The mathematical justification for the DICs theory takes its foundations in the theory of singular perturbations (Fenichel, 1979). Current variations can be described at different timescales, where each scale is assumed to be *quasi-static* relative to the others. This means that faster scales are already at equilibrium, while slower scales exhibit negligible variations:

$$\Delta I = \Delta I_f + \Delta I_s + \Delta I_u \tag{2.8}$$

The quasi-static assumption allows us to apply an approach similar to equation (2.6) for each timescale separately:

$$\begin{cases} \Delta I_f & \approx -g_f(V)\Delta V \\ \Delta I_s & \approx -g_s(V)\Delta V \\ \Delta I_u & \approx -g_u(V)\Delta V \end{cases} \overset{\Delta I \approx -g_t(V)\Delta V}{\Longrightarrow} g_f(V) + g_s(V) + g_u(V) = g_t(V) \tag{2.9}$$

Thus, the total DIC curve $g_t(V)$ can be decomposed into a fast dynamic conductance curve $g_f(V)$, a slow dynamic conductance curve $g_s(V)$, and an ultra-slow dynamic conductance curve $g_u(V)$. Each of these curves reflects the voltage-current relationship within a specific timescale:

$$g_j(V) = -\left(\frac{\partial I_j}{\partial V}\right)^{-1}, \quad j \in \{f, s, u\} \tag{2.10}$$

While DICs can be measured experimentally, they ca also be constructed analytically for CBMs. By slightly adapting the definition from Drion et al., 2015, we obtain:

$$\begin{aligned}
g_f(V) &= -\frac{\partial I_f}{\partial V}(V) g_{\text{leak}}^{-1} = \left[-\frac{\partial \dot{V}}{\partial V} - \sum_i w_{fs,X_i}\left(\frac{\dot{V}}{\partial X_i}\frac{\partial X_{i,\infty}}{\partial V}\right)\right]\Bigg|_V \frac{1}{g_{\text{leak}}}, \\
g_s(V) &= -\frac{\partial I_s}{\partial V}(V) g_{\text{leak}}^{-1} = \left[-\sum_i \left(w_{su,X_i} - w_{fs,X_i}\right)\left(\frac{\dot{V}}{\partial X_i}\frac{\partial X_{i,\infty}}{\partial V}\right)\right]\Bigg|_V \frac{1}{g_{\text{leak}}}, \\
g_u(V) &= -\frac{\partial I_u}{\partial V}(V) g_{\text{leak}}^{-1} = \left[-\sum_i \left(1 - w_{su,X_i}\right)\left(\frac{\dot{V}}{\partial X_i}\frac{\partial X_{i,\infty}}{\partial V}\right)\right]\Bigg|_V \frac{1}{g_{\text{leak}}}, \\
g_t(V) &= g_f(V) + g_s(V) + g_u(V),
\end{aligned} \tag{2.11}$$

where:

- The terms X_i correspond to either (a) **Gating variables** (m_i or h_i) that control the activation and inactivation of ion channels; or (b) **Ion modeling variables**, such as Ca in the STG model, which represent intracellular ion concentrations influencing channel dynamics.

- The terms $w_{fs,X_i}(V)$ and $w_{su,X_i}(V)$ are **voltage-dependent weighting factors**.

These weights are central to DICs theory, as they determine how each variable contributes to different timescales. Their values are computed based on a logarithmic scaling between 0 and 1, using reference timescales, as defined in equation (2.12):

$$w_{\text{fs},X_i}(V) = \begin{cases} 1 & , \quad \tau_{X_i}(V) \leq \tau_\text{f}(V) \\ \dfrac{\log\left(\tau_\text{s}(V)\right) - \log\left(\tau_{X_i}(V)\right)}{\log\left(\tau_\text{s}(V)\right) - \log\left(\tau_\text{f}(V)\right)} & , \quad \tau_\text{f}(V) < \tau_{X_i}(V) \leq \tau_\text{s}(V) \\ 0 & , \quad \tau_{X_i}(V) > \tau_\text{s}(V) \end{cases}$$
$$w_{\text{su},X_i}(V) = \begin{cases} 1 & , \quad \tau_{X_i}(V) \leq \tau_\text{s}(V) \\ \dfrac{\log\left(\tau_\text{u}(V)\right) - \log\left(\tau_{X_i}(V)\right)}{\log\left(\tau_\text{u}(V)\right) - \log\left(\tau_\text{s}(V)\right)} & , \quad \tau_\text{s}(V) < \tau_{X_i}(V) \leq \tau_\text{u}(V) \\ 0 & , \quad \tau_{X_i}(V) > \tau_\text{u}(V) \end{cases} \tag{2.12}$$

The reference timescales τ_f, τ_s, and τ_u are chosen according to the characteristic timescales of bursting and spiking dynamics. τ_f corresponds to the activation time constant of the fastest depolarizing current, while τ_s represents the activation time constant of the fastest repolarizing current. Finally, τ_u is associated with the slowest variable in the system, which typically governs burst adaptation.

For the STG model, these timescales correspond to $\tau_{m_\text{Na}}(V)$, $\tau_{m_\text{Kd}}(V)$, and $\tau_\text{H}(V)$. In the DA neuron model, they are given by $\tau_{m_\text{Na}}(V)$, $\tau_{m_\text{Kd}}(V)$, and a constant-value function $\tau_\text{u,DA} = 100\,\text{ms}$. Finally, in the standard case of purely voltage-dependent ion channels, the factors $\frac{\partial \dot{V}}{\partial X_i}$ can be expressed as:

$$\frac{\partial \dot{V}}{\partial m_i} = \bar{g}_i p_i m_{i,\infty}^{p_i - 1} h_{i,\infty}^{q_i}; \qquad \frac{\partial \dot{V}}{\partial h_i} = \bar{g}_i q_i m_{i,\infty}^{p_i} h_{i,\infty}^{q_i - 1},$$

For cases involving ionic dependence with explicit modeling, as in the STG model, additional ionic partial derivatives must be included [6], e.g., $\frac{\partial \dot{V}}{\partial \text{Ca}} \frac{\partial \text{Ca}_\infty}{\partial V}$ for the STG model.

We emphasize here that the analytical nature of CBM DICs is a real strength of this theory, as it allows us to get around the problem of CBM intractability: we do not need to simulate the model to compute its DICs.

Each conductance contributes to multiple timescales A fundamental aspect of understanding DICs theory is the temporal separation of conductances. One important point is that a single ion channel can influence multiple timescales. Mathematically, this is reflected in the dependence of the weighting factors $w_{\text{fs},X_i}(V)$ and $w_{\text{su},X_i}(V)$ on voltage. Since we focus on *dynamic* behaviors, the membrane potential evolves over time, meaning the distribution of conductances across timescales also changes. Moreover, a current can have rapid activation and slow inactivation, accentuating the fact that the same current can belong to several timescales.

Although it is common in the literature to classify some currents as "fast" or "slow" — or even in the presentation of the STG model in Section 2.2.2 of this work — this is, in practice, a simplification. The impact of each current on different timescales is dynamic.

Figure 2.11 illustrates the values of the weighting factors for the STG model, highlighting the variable distribution of conductances across timescales. Thus, the description of neuronal dynamics using only three timescales is a characteristic of the emergent behavior of neurons, rather than a strict property of the individual ion currents themselves.

[6] It is to make these mathematically manageable that the dependence of E_Ca on the value of Ca of the STG model is neglected in this work, in contrast with the original model of Liu et al., 1998. See Appendix B.1.

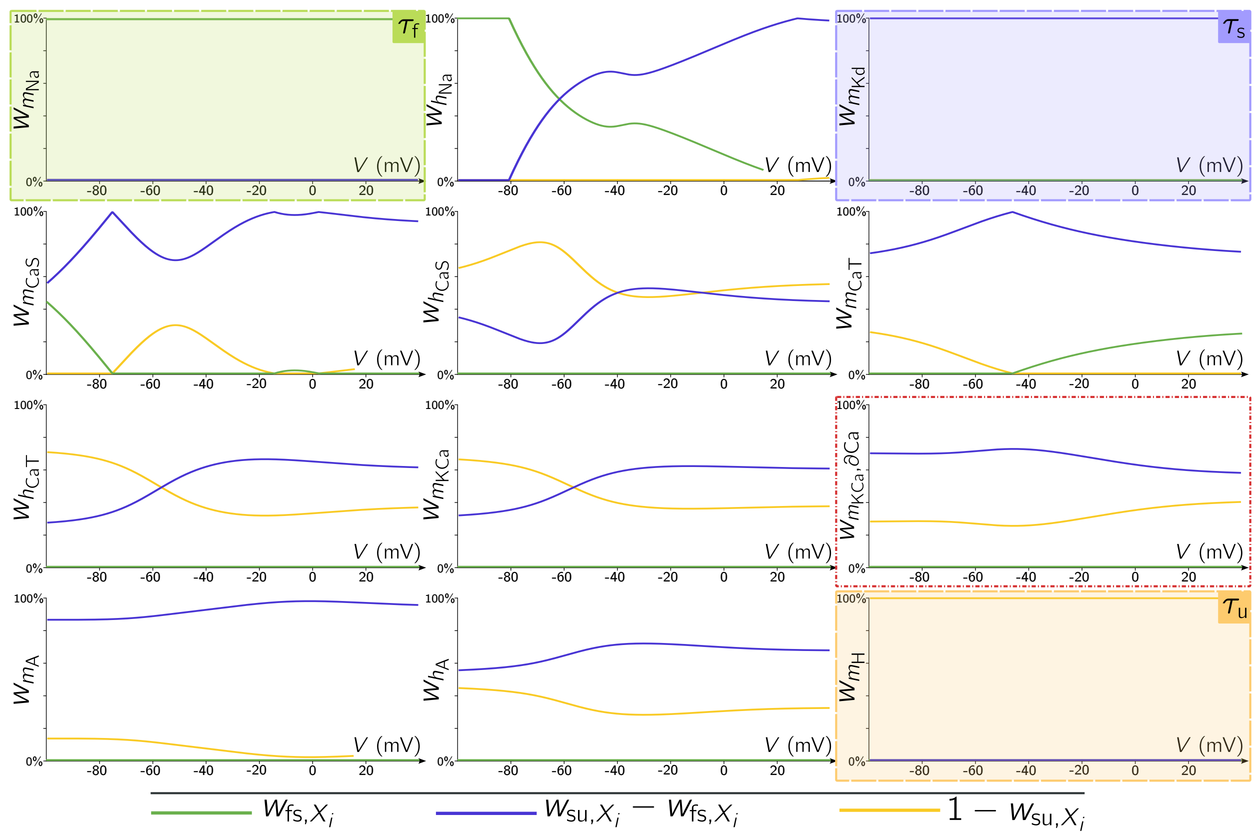


Figure 2.11: Voltage-dependent weighting factors for the fast, slow, and ultra-slow timescales in the STG model. Green, purple, and yellow represent the contributions to the fast, slow, and ultra-slow timescales, respectively. Boxed panels (except for the red one) highlight the reference gating variable for each timescale. The distribution of weighting factors varies with voltage, demonstrating how individual conductance contributes dynamically across multiple timescales. The red boxed panel is special, as it corresponds to calcium dynamics specific to the STG model. Unlike the other panels, it does not represent the weighting factor of a gating variable directly but rather accounts for the influence of intracellular calcium dynamics.

The threshold voltage This is a critical concept in the DIC framework, marking the membrane potential at which neuronal sensitivity to parameters perturbations is maximized. At this voltage, even small changes in parameters can significantly influence neuronal response, making V_{th} a pivotal parameter in excitability shaping.

Given the DICs framework, a convenient voltage threshold definition is that V_{th} corresponds to the membrane potential where the system net feedback is at a critical balance, maximizing neuronal sensitivity. This balance arises from the interaction between fast, slow, and ultra-slow dynamic conductances, which collectively create a dynamic equilibrium between stabilizing (negative feedback) and destabilizing (positive feedback) forces.

The exact value of V_{th} is instance-dependent, influenced by the specific composition of ion channel conductances in a given neuron. The original work by Franci et al., 2013 proposes to identify the threshold potential as a transcritical bifurcation. More recent work such as Fyon et al., 2024 explains that a more direct alternative method is to identify the first root of g_t around which g_t is decreasing (so we switch from a net negative feedback to a net positive feedback) as an approximation of V_{th}. For the sake of simplicity, we'll refer to this as a '*decreasing*' zero. That is :

$$g_t(V_{th}) = 0 \quad \text{with} \quad g_t(V_{th} - \delta V) > 0 > g_t(V_{th} + \delta V), \quad \forall \text{ sufficiently small } \delta V > 0 \tag{2.13}$$

Even if the exact value of V_{th} is not instance-independent, we can see (Fig. 2.10, bottom right) that they are closely distributed around a model-dependent instance-independent value. This value will be of great importance in the following of this work and will be evaluated for the STG model and the DA model. Knowing the exact value of V_{th} is not necessary to apply the DICs theory, but it's observing the values of DICs, around this value that shapes neuronal activity such that we need to be close enough. In the rest of this work, we'll refer to this model-dependent instance-independent value when we talk about V_{th}, rather than the precise instance-dependent value.

DICs at the threshold voltage shape the neuronal activity The central reason DICs theory is well suited to the problems addressed in this thesis lies in the direct link between spontaneous neuronal activity and the values of DICs around V_{th}. An intuition for this phenomenon can be gained by examining Figures 2.10 and 2.12.

First, Figure 2.10 shows that different neuronal activity patterns correspond to distinct DIC curves, particularly differing at V_{th}. Next, Figure 2.12 presents three degenerate instances: although their overall DIC curves differ because their maximal conductances sets differ, their values at V_{th} are identical and their overall spontaneous neuronal activities are similar.

In other words, degeneracy arises because multiple different $\bar{\boldsymbol{g}}$ configurations can lead to the same $\boldsymbol{g}_{\mathrm{DICs}}(V_{\mathrm{th}})$. Consequently, this work focuses on DICs theory at the threshold voltage, using the notation $\boldsymbol{g}_{\mathrm{DICs}} = \boldsymbol{g}_{\mathrm{DICs}}(V_{\mathrm{th}})$. More specifically, we are interested in slow and ultra-slow dynamic conductances, as these are what shape the spike times sequence. In other words, although three timescales are necessary to describe the activity, the fast dynamic conductance is of no quantitative interest, provided it imposes a sufficiently large positive feedback around the threshold.

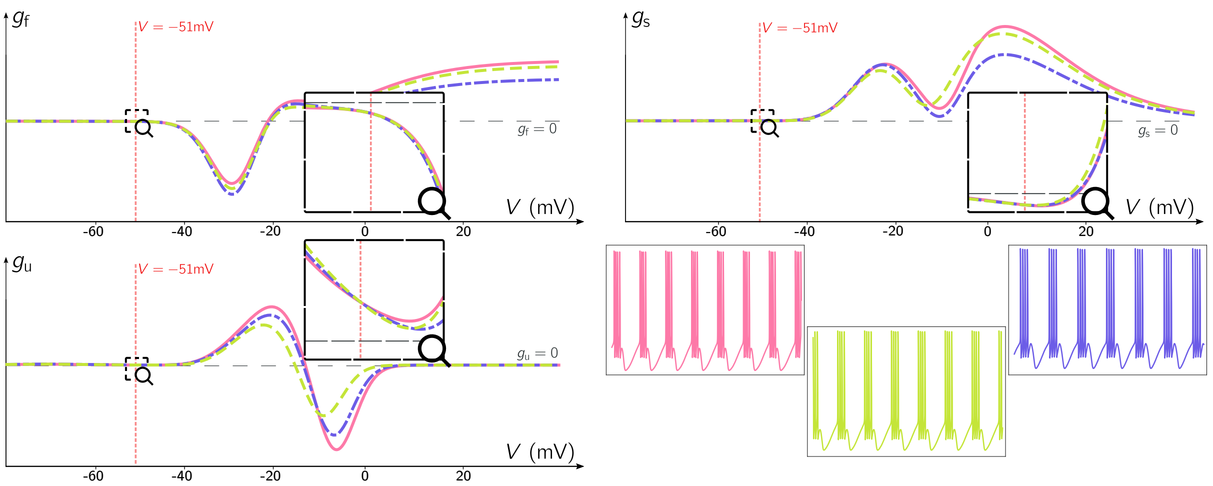


Figure 2.12: Degeneracy in DIC curves. We compare three degenerate instances of the STG model exhibiting distinct DIC curves but identical values at the threshold voltage. Each panel represents one of the three timescale-specific conductance components: g_f (fast), g_s (slow), and g_u (ultra-slow). While the overall DIC curves differ between instances (because their maximal conductance values also differ), they converge at the threshold voltage V_{th}, highlighted by the vertical red dashed line. This shared value of $\boldsymbol{g}_{\mathrm{DICs}}(V_{\mathrm{th}})$ among different instances explains how different conductance compositions can lead to similar neuronal dynamics.

The compensation algorithm The mathematical framework provided by DICs theory allows for the identification of a simple compensation procedure. Compensation refers to a set of parametric variations that preserve neuronal activity. Intuitively, the compensation algorithm focuses on maintaining the DIC values at the threshold potential V_{th}, ensuring that changes in individual conductances do not disrupt the overall neuronal activity.

In practice, the algorithm is based on the *sensitivity* of DICs to different maximal conductances:

$$S_{ij} = \frac{\partial g_{\mathrm{DICs},i}}{\partial \bar{g}_j} \implies \Delta g_{\mathrm{DICs},i} \approx \Delta \bar{g}_j S_{ij}, \quad (i,j) \in \{\mathrm{f}, \mathrm{s}, \mathrm{u}\} \times \{1, \dots, N_{\mathrm{model}}\} \tag{2.14}$$

This equation expresses how small changes in a given conductance $\bar{g}_j$ affect the DICs across different timescales. When a set of conductances is perturbed, compensation is achieved by adjusting another set of conductances with opposite sensitivity.

In the simplest cases, the compensation algorithm can be formulated as a linear system:

$$\underbrace{\boldsymbol{S}_{\mathrm{comp.}}}_{A}\underbrace{\Delta\bar{\boldsymbol{g}}_{\mathrm{comp.}}}_{x} = \underbrace{-\boldsymbol{S}_{\mathrm{perturb.}}\Delta\bar{\boldsymbol{g}}_{\mathrm{perturb.}}}_{b} \Leftrightarrow \underbrace{\boldsymbol{S}_{\mathrm{comp.}}}_{A}\underbrace{\bar{\boldsymbol{g}}_{\mathrm{comp.}}}_{x} = \underbrace{\boldsymbol{g}_{\mathrm{DICs}} - \boldsymbol{S}_{\neg\mathrm{comp.}}\bar{\boldsymbol{g}}_{\neg\mathrm{comp.}}}_{b} \tag{2.15}$$

where:

- “comp.” refers to the set of conductances used for compensation, and “perturb.” refers to the conductances that were perturbed.

- The notation “$\neg$ comp.” (not compensated) refers to all conductances in the CBM except those actively used for compensation. This includes the perturbed conductances.

- $\Delta\bar{\boldsymbol{g}}_{\mathrm{set}}$ represents the changes in conductance values after perturbation and compensation.

- $\boldsymbol{S}_{\mathrm{set}}$ is the **sensitivity matrix**, which plays a central role in this method. Formally, for a given set:

$$\bar{\boldsymbol{g}}_{\mathrm{set}} \in \mathbb{R}_{+}^{N_{\mathrm{set}}} \Leftrightarrow \boldsymbol{S}_{\mathrm{set}} \in \mathbb{R}^{3\times N_{\mathrm{set}}} \tag{2.16}$$

Each row of $\boldsymbol{S}_{\mathrm{set}}$ corresponds to one of the three timescales (fast, slow, ultra-slow), while each column corresponds to a specific conductance. The product $\boldsymbol{S}_{\mathrm{set}}\bar{\boldsymbol{g}}_{\mathrm{set}}$ represents a partial sum of the definition of DICs, with the passive term decomposed per conductance in the fast timescale. In the simplest cases, $\boldsymbol{S}_{\mathrm{set}}\Delta\bar{\boldsymbol{g}}_{\mathrm{set}} \approx \Delta\boldsymbol{g}_{\mathrm{DICs}}$.

Compensation is feasible when functional overlap exists, meaning there is sufficient sensitivity across the relevant timescales. Figure 2.13 provides an example of the sensitivity matrix for the STG model. Sensitivity depends on multiple factors, including weight functions, Nernst potentials, and steady-state gating functions. However, for most conductances, the sensitivity evaluated at V_{th} remains constant between different instances. An exception is the sensitivity of $\bar{g}_{\mathrm{KCa}}$, which is instance-dependent due to its dependence on intracellular calcium, whose steady-state value is influenced by the specific conductances $\bar{\boldsymbol{g}}$.

The sensitivity matrix reveals that some conductances have positive sensitivities while others have negative ones, spanning several orders of magnitude. Additionally, sensitivity varies across timescales, reinforcing the importance of a timescale-dependent compensation approach.

However, while the DICs theory effectively reduces the complexity of CBMs, it also raises an important question: how can we reconstruct one or several degenerate CBMs from the DIC values? In other words, how can we invert the relation described in equation (2.7) to generate a degenerate population of CBMs? This question is fundamental to applying the DICs theory to our work and will be addressed in the following section. A summary of important concepts of DICs theory is available in Box IV.

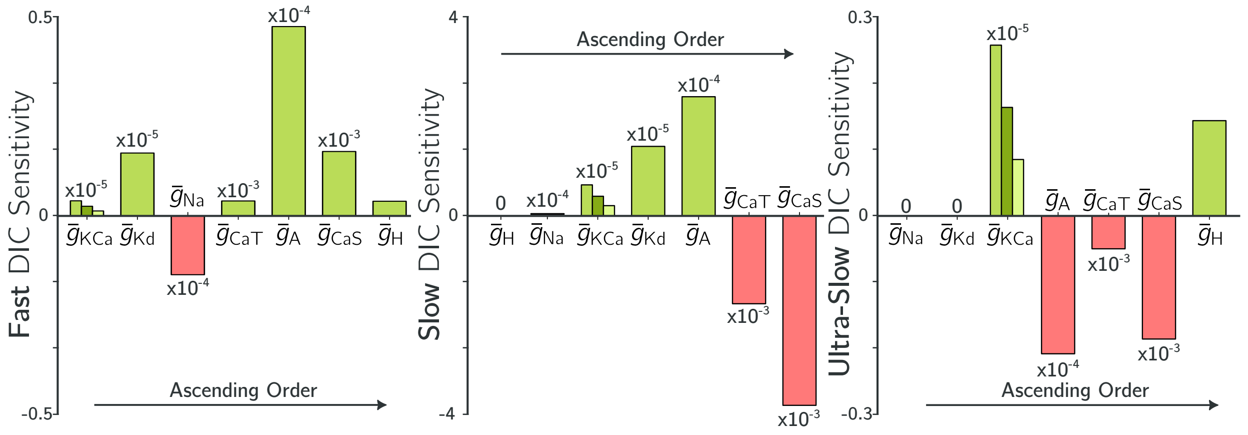


Figure 2.13: Sensitivity of DIC components to maximal conductances at the threshold voltage in the STG model. Each panel represents the sensitivity of DICs *at the threshold voltage* for different timescales: fast (left), slow (middle), and ultra-slow (right). Conductances are arranged in ascending order of sensitivity magnitude from left to right. Green bars indicate positive sensitivity, while red bars indicate negative sensitivity. The $\bar{g}_{\mathrm{KCa}}$ conductance appears three times, corresponding to three distinct model instances, as its sensitivity depends on intracellular calcium dynamics. The sensitivity matrix spans several orders of magnitude and varies across timescales, highlighting the need for a timescale-dependent compensation approach. Functional overlap between conductances enables compensation by ensuring sufficient sensitivity across timescales.

Box IV – Reducing Dimensionality and Handling Degeneracy with Dynamic Input Conductances

The Theory of Dynamic Input Conductances (DICs)

Dynamic Input Conductances (DICs) provide a mathematical framework for analyzing the effects of parameter variations on conductance-based models (CBMs). This approach reduces the complexity of high-dimensional parameter spaces and addresses the issue of degeneracy by decomposing membrane dynamics into distinct timescales. The key aspects of DICs are:

- **Decomposition of Membrane Dynamics:** Membrane dynamics are decomposed into three distinct timescales through a *quasi-static assumption*: **fast** ($g_\mathrm{f}(V)$) for spike upstroke, **slow** ($g_\mathrm{s}(V)$) for spike downstroke and interspike period, and **ultra-slow** ($g_\mathrm{u}(V)$) for spike adaptation and interburst intervals.
- **Dynamic Input Conductances:** The total conductance $g_\mathrm{t}(V)$ is decomposed into these three components: $g_\mathrm{t}(V) = g_\mathrm{f}(V) + g_\mathrm{s}(V) + g_\mathrm{u}(V)$, where each $g_j(V)$ is defined as:

$$g_j(V) = -\left(\frac{\partial I_j}{\partial V}\right)^{-1}, \quad j \in \{\mathrm{f}, \mathrm{s}, \mathrm{u}\}.$$

- **Weighting Factors:** Voltage-dependent weighting factors $w_{\mathrm{fs},X_i}(V)$ and $w_{\mathrm{su},X_i}(V)$ determine the contribution of each variable X_i to the different timescales. They can be expressed via the Eq. (2.12). Combined with Eq. (2.11), we are able to compute the DICs of any CBMs.
- **Threshold Voltage:** The threshold voltage, V_th, marks the membrane potential at which neuronal sensitivity to parameter perturbations is maximized. We can find it using Eq. (2.13).
- **Compensation Algorithm:** Thanks to the functional overlapping of conductances in CBMs, it is possible to identify sets of $\bar{\boldsymbol{g}}$ variations that do not impact the spontaneous activity of the system. In the simplest cases, the compensation procedure can be written as in Eq. (2.15).

The Theory of DICs in this work

In this work, the DICs theory is used, as it is assumed to encode neuronal activity completely by the value of the DICs around the threshold potential, $\boldsymbol{g}_\mathrm{DICs}$. We are therefore able to use it to identify a very wide range of maximal conductance configurations related to recorded neuronal activity.

$$\underbrace{\bar{\boldsymbol{g}} \in \mathbb{R}_+^{N_\mathrm{model}} \xrightarrow{\text{Dynamic Input Conductances}} \boldsymbol{g}_\mathrm{DICs}(V) = \begin{bmatrix} g_\mathrm{f}(V) \\ g_\mathrm{s}(V) \\ g_\mathrm{u}(V) \end{bmatrix} : \mathbb{R} \to \mathbb{R}^3 \xrightarrow{\text{Threshold Potential } V_\mathrm{th}} \boldsymbol{g}_\mathrm{DICs}(V_\mathrm{th}) \in \mathbb{R}^3}_{\texttt{Reducing the dimensionality of CBMs through DICs, from } \mathbb{R}_+^{N_\mathrm{model}} \texttt{ to } \mathbb{R}^3.}$$

2.4 An Efficient Algorithm to Generate Degenerate Populations

The previous section explained how the DICs theory can be used to reduce the description of neuronal activity to only three values, via the relation (2.7). However, it does not introduce a fundamental component necessary to make the DICs theory applicable to our work: how, from the DIC values, can one reconstruct one or several degenerate CBMs? In other words, how can we invert the relation (2.7):

$$\underbrace{\boldsymbol{g}_{\text{DICs}} \in \mathbb{R}^3 \xrightarrow{\text{Generate a Degenerate Population}} \bar{\boldsymbol{g}} \in \mathbb{R}_+^{N_{\text{model}}}}_{\texttt{Augmenting back the DICs at threshold potential into several degenerate instances of CBMs.}} \tag{2.17}$$

2.4.1 Preliminary Considerations

In a recent work, Fyon et al., 2024 proposed an *efficient* algorithm for generating degenerate populations, ensuring precise control over the effects of *homogeneous scaling* and *variability in conductance ratios*.

Two mechanisms underlying neuronal degeneracy These two effects play a fundamental role in understanding degeneracy in CBMs. The first, homogeneous scaling, corresponds to a global and proportional scaling of all conductances in a given model. This transformation can be formalized as:

$$\bar{\boldsymbol{g}} \xrightarrow{\text{Homogeneous scaling}} \alpha\bar{\boldsymbol{g}}, \quad \alpha \in \mathbb{R}^+$$

In other words, homogeneous scaling uniformly modifies all conductances without changing their relative ratios, thereby preserving the intrinsic dynamics of the model *while influencing its response to external inputs*. Homogeneous scaling directly affects the neuron input resistance.

The second effect, variability in conductance ratios, corresponds to a non-proportional variation in the relationships between different conductances in a model. This transformation can be expressed as:

$$\bar{\boldsymbol{g}} \xrightarrow{\text{Variability in conductance ratios}} \bar{\boldsymbol{g}} + \Delta\bar{\boldsymbol{g}}, \quad \Delta\bar{\boldsymbol{g}} \in \mathbb{R}^{N_{\text{model}}}$$

where $\Delta\bar{\boldsymbol{g}}$ represents a non-proportional modification of conductances, altering their interdependencies. The exact components of $\Delta\bar{\boldsymbol{g}}$ are not independent, however, so that the different variations compensate for each other. Unlike homogeneous scaling, variability in conductance ratios has almost no effect on the neuron input resistance.

Thus, the generation of a degenerate population can be achieved through the controlled combination of these two effects.

A computationally efficient algorithm The efficiency of the method proposed by Fyon et al., 2024 lies in its ability to generate a degenerate population directly, without requiring post-selection or filtering of individual models. Prior to this approach, exploring the space of maximal conductances required a blind search — i.e., naively sampling this space, simulating each individual, and retaining only those that exhibited the desired behavior. This approach relied on a tedious trial-and-error process. In contrast, the proposed method produces a population that is *a priori* degenerate, eliminating the need for a filtering step and even simulation of the instances; *making the process significantly faster*.

Conceptually, the algorithm relies on the use of the *compensation algorithm* derived from DICs theory to adjust specific conductances after an initial blind sampling. Since the DIC values at the threshold potential largely determine neuronal activity and the method constrains these values, the resulting population is indeed degenerate. Algorithm 1 presents a more detailed pseudo-code version of the algorithm introduced in Fyon et al., 2024.

More precisely, the generation procedure is divided into two distinct steps. The first step consists of generating a degenerate population exhibiting spontaneous spiking, a crucial phase in this work, which focuses on degenerate populations displaying spontaneous bursting or spiking activity. This initial step is achieved by enforcing a strongly negative value of g_f, in accordance with the DICs theory presented in Section 2.3. In fact, a very negative g_f value indicates a large amount of fast positive feedback encouraging the initiation of an action potential and therefore spontaneous activity.

The second step involves modulating this spontaneous population to impose the desired values of g_s and g_u — that is, dynamic input conductances over the slow and ultra-slow timescales, as these two parameters directly shape the neuronal spontaneous activity according to the DICs theory.

Formally, the first step consists of dividing the set of model conductances into two groups:

$$\bar{\boldsymbol{g}} \in \mathbb{R}^N \to \bar{\boldsymbol{g}}_\mathrm{random} \in \mathbb{R}^{N-n} \quad \text{and} \quad \bar{\boldsymbol{g}}_\mathrm{comp.,\ spont.} \in \mathbb{R}^n.$$

Using these notations, n represents the number of conductances that need to be compensated to generate a spontaneous population. Here, $n = 3$, corresponding to the number of distinct timescales used in DICs theory.

The group $\bar{\boldsymbol{g}}_\mathrm{random}$ is sampled from a distribution $\mathcal{D}$, which must be chosen *a priori* by the experimentalist or the modeler. This distribution plays a role in shaping the degeneracy of the population generated by the algorithm, and it directly relates to the *level of degeneracy*. The remaining conductances, grouped as $\bar{\boldsymbol{g}}_\mathrm{comp.,\ spont.}$, are computed so that the DIC values at the threshold potential V_th satisfy: $\boldsymbol{g}_\mathrm{DICs,spont.} \in \mathbb{R}^n$.

This ensures that, at the end of this step, the generated population exhibits spontaneous activity. Determining the remaining conductances to impose specific DIC values is the core of the *compensation algorithm* in DICs theory. In practice, this is done by constructing the sensitivity matrix $\boldsymbol{S} \in \mathbb{R}^{3\times N_\mathrm{model}}$, which is then partitioned as follows, consistently with conductances separation:

$$\boldsymbol{S} \to \boldsymbol{S}_\mathrm{random} \in \mathbb{R}^{n\times N-n} \quad \text{and} \quad \boldsymbol{S}_\mathrm{comp.,\ spont.} \in \mathbb{R}^{n\times n}$$

A linear system can then be formulated and solved to determine the values of $\bar{\boldsymbol{g}}_\mathrm{comp.,\ spont.}$, following Eq. (2.15) and with the random set corresponding to the not compensated set.

The second step is similar to the first, but instead of randomly sampling conductances, the distribution obtained at the end of the first step is used as a starting point. A different partitioning of $\bar{\boldsymbol{g}}$ is performed:

$$\bar{\boldsymbol{g}} \to \bar{\boldsymbol{g}}_\text{random from step 1} \in \mathbb{R}^{N-n+1} \quad \text{and} \quad \bar{\boldsymbol{g}}_\mathrm{comp.} \in \mathbb{R}^{n-1}.$$

The same compensation process is then applied to impose the desired values of $\boldsymbol{g}_\mathrm{DICs} \in \mathbb{R}^{n-1}$. During this step, only $n-1$ of the DIC values are enforced. Specifically, no new constraint is imposed on g_f. This is because, at the end of the first step, g_f is already sufficiently negative to ensure that the population exhibits spontaneous activity. The other timescales (g_s and g_u) primarily determine the specific properties of the latter.

In both steps, the choice of conductances to be compensated is not entirely free. The linear system must have a valid solution, meaning that the matrices $\boldsymbol{S}_\mathrm{comp.}$ and $\boldsymbol{S}_\mathrm{comp.,\ spont.}$ must be non-singular. From a DICs theory perspective, this implies that the compensated conductances must, collectively, have an impact on the relevant timescales. For example, it is not possible to control the $g_\mathrm{s}(V)$ and $g_\mathrm{u}(V)$ curves using only $\bar{g}_\mathrm{Na}$ and $\bar{g}_\mathrm{Kd}$, because neither of these conductances influences the *ultra-slow* timescale as it can be seen from the sensitivity values in Figure 2.13.

In Fyon et al., 2024, the authors explain how to construct $\mathcal{D}$ in order to eliminate the effect of *homogeneous scaling*. The idea is to first sample g_leak from a distribution $\mathcal{D}_{g_\mathrm{leak}}$, then sample the other maximal conductances and normalize them by the factor $\frac{g_\mathrm{leak}}{g_\mathrm{leak,\ mean}}$, where $g_\mathrm{leak,\ mean}$ is the mean of the distribution $\mathcal{D}_{g_\mathrm{leak}}$. Statistically, this transformation imposes a linear dependency between the maximal conductances and g_{leak}, thereby introducing a specific covariance structure that eliminates the effect of *homogeneous scaling*.

In practice, the algorithm is used to generate degenerate populations without homogeneous scaling, and this effect is later reintroduced via the leak conductance, allowing controlled modulation of input resistance while preserving the intrinsic dynamics of the population.

Algorithm 1: Efficient Generation of Degenerate Populations with Spontaneous Activity

```
Data:
    - The size of the population: $M$
    - Conductance distribution $\mathcal{D}$: $\bar{\boldsymbol{g}}_{\text{random}} \sim \mathcal{D}$
    - Threshold potential $V_{\text{th}}$
    - Target DIC values at threshold: $\boldsymbol{g}_{\text{DICs}}$
    - Conductances to be compensated: $\bar{\boldsymbol{g}}_{\text{comp.}} \subseteq \bar{\boldsymbol{g}}$
    - Target DIC values for spontaneous spiking: $\boldsymbol{g}_{\text{DICs, spont.}}$
    - Conductances to be compensated for spontaneous spiking: $\bar{\boldsymbol{g}}_{\text{comp., spont.}} \subseteq \bar{\boldsymbol{g}}$
Result: A degenerate population exhibiting spontaneous activity
for $i \leftarrow 1$ to $M$ do
    // Step 1:  Generate the i-th individual with spontaneous spiking
    Sample $\bar{\boldsymbol{g}} \sim \mathcal{D}$
    // Construct sensitivity matrices
    Compute $\boldsymbol{S}(V_{\text{th}})$, the sensitivity matrix associated with DICs at threshold
    Extract $\boldsymbol{S}_{\text{comp., spont.}}$, the submatrix corresponding to compensated conductances
    Extract $\boldsymbol{S}_{\text{random}}$, the submatrix corresponding to randomly sampled conductances
    // Adjust conductances for spontaneous spiking
    Solve the system:
        $\boldsymbol{S}_{\text{comp., spont.}} \cdot \bar{\boldsymbol{g}}_{\text{comp., spont.}} = \boldsymbol{g}_{\text{DICs, spont.}} - \boldsymbol{S}_{\text{random}} \cdot \bar{\boldsymbol{g}}_{\text{random}}$
    // Step 2:  Modulate the individual to refine spontaneous activity
    // Construct sensitivity matrices for this step
    Extract $\boldsymbol{S}_{\text{comp.}}$ and $\boldsymbol{S}_{\text{random}}$
    // Adjust conductances for target DIC values
    Solve the system:
        $\boldsymbol{S}_{\text{comp.}} \cdot \bar{\boldsymbol{g}}_{\text{comp.}} = \boldsymbol{g}_{\text{DICs}} - \boldsymbol{S}_{\text{random}} \cdot \bar{\boldsymbol{g}}_{\text{random}}$
    Store $\bar{\boldsymbol{g}}$ in the population
return Degenerate population $\{\bar{\boldsymbol{g}}_i\}_{i=1}^{M}$ with targeted spontaneous activity
```

A concrete example and non-linearity in conductance compensation For example, following Fyon et al., 2024, it is possible to generate degenerate populations from the STG model by compensating the conductances $\bar{g}_{\text{A}}$ and $\bar{g}_{\text{CaS}}$. All the details are avaible in Appendix E.

Figures 2.14A, 2.14B, and 2.14C illustrate the degenerate populations obtained in strong bursting, light bursting, and tonic spiking modes, respectively. It can be observed that, although individuals within the same population exhibit globally similar activity, their maximal conductance values vary significantly — sometimes by several orders of magnitude, as one would observe in biology.

2.4.2 Critical Insights Leading to Major Contributions

These examples highlight several important aspects regarding degenerate populations and Algorithm 1.

First, although these populations are classified as degenerate, a certain degree of activity variability remains among the individuals composing them. This variability manifests as slight differences in spiking or bursting frequencies around the mean value characterizing the degenerate population activity. In this work, we consider this variability negligible and assume, in general, that populations can be treated as degenerate ensembles exhibiting homogeneous activity. In other words, we consider that the *degenerative proximity* inherent from the $\boldsymbol{g}_{\text{DICs}}$ is sufficient.

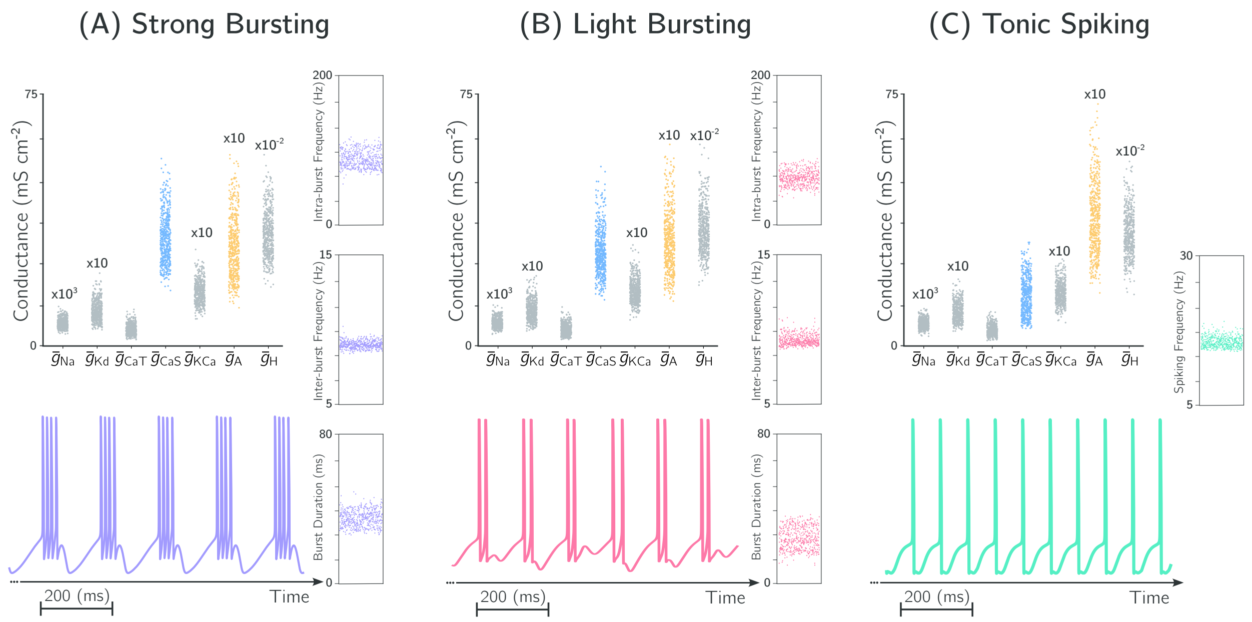


Figure 2.14: Illustration of degenerate populations generated using Algorithm 1. The figure presents three panels (A, B, and C), each corresponding to a population of $M = 500$ instances classified as degenerate. The three panels represent distinct activity modes: strong bursting (A, purple), light bursting (B, pink), and tonic spiking (C, cyan). In each case, all instances share the same DICs value at the threshold, ensuring activity similarity even though conductance values vary. The top subplot in each panel shows the distribution of maximal conductances across the population. Conductances in gray correspond to non-compensated values, while the colored conductances ($\bar{g}_{\mathrm{CaS}}$ and $\bar{g}_{\mathrm{A}}$) indicate the parameters that were adjusted to maintain the expected activity patterns, following the compensation method of Fyon et al., 2024. The values are displayed as scaled versions of the true conductances, with the appropriate scaling factor indicated above each column. The middle subplot displays a representative voltage trace from the population, illustrating the expected firing behavior. The right subplot presents the distribution of key activity metrics within the population, revealing a degree of variability among individual instances. However, we assume that this variability remains within the bounds of degenerate activity. For bursting populations (A, B), the metrics include (from bottom to top) burst duration (ms), inter-burst frequency (Hz), and intra-burst frequency (Hz). For the spiking population (C), the metric shown is the spiking frequency. Each point represents an individual instance within the population.

This slight variability underscores a crucial point: while DICs provide a powerful tool, enforcing the values of the $g_{\mathrm{s}}(V)$ and $g_{\mathrm{u}}(V)$ curves at a single point still allows for some variability. However, increasing the number of constrained points or introducing additional timescales would reduce the *level of degeneracy* permitted. Moreover, slight variability is consistent with biology, which is inherently noisy.

Additionally, the generation procedure described in Algorithm 1 is, in fact, a simplification and an approximation, in the sense that the results obtained are not exactly as theoretically predicted. This is particularly true in the context of the STG model when choosing to compensate for the conductance $\bar{g}_{\mathrm{CaS}}$, as the imposed values of $\boldsymbol{g}_{\mathrm{DICs}}$ are not accurately enforced.

Indeed, what the procedure described here does not explicitly show is that when compensating for the conductances $\bar{g}_{\mathrm{CaS}}$ or $\bar{g}_{\mathrm{CaT}}$ in the STG model, the compensation problem can not be reduced to solving a linear system. In practice, the sensitivity matrix $\boldsymbol{S}$ is dependent on both voltage and calcium concentration, i.e., $\boldsymbol{S} = \boldsymbol{S}(V, \mathrm{Ca})$, and should be evaluated at the equilibrium value $\mathrm{Ca} = \mathrm{Ca}_{\infty}(\bar{\boldsymbol{g}}, V)$.

This dependency of $\boldsymbol{S}$ on internal system variables beyond V is expected in all CBMs that incorporate an explicit ionic dynamics model. Conversely, while in the DA model some gating variables depend on magnesium, the dynamics of Mg are not explicitly modeled. Consequently, in this case, $\boldsymbol{S}$ does depend on

Mg but there is no $\mathrm{Mg} = \mathrm{Mg}_\infty(\cdot)$ to compute. For models similar to the DA model used in this work — i.e., those where $\boldsymbol{S}$ depends directly only on V — Algorithm 1 remains exact.

In the case of the STG model, the system is no longer linear because, although the calcium equilibrium value depends linearly on $\bar{g}_{\mathrm{CaS}}$ and $\bar{g}_{\mathrm{CaT}}$ (2.18), the dependence of $\boldsymbol{S}$ on calcium itself is not linear (2.19):

$$\dot{\mathrm{Ca}}\Big|_{\mathrm{Ca}=\mathrm{Ca}_\infty} = 0 \implies \mathrm{Ca}_\infty = -\alpha_{\mathrm{Ca}}\left(\bar{g}_{\mathrm{CaT}} m^3_{\mathrm{CaT},\infty}(V_{\mathrm{th}}) h_{\mathrm{CaT},\infty}(V_{\mathrm{th}}) + \bar{g}_{\mathrm{CaS}} m^3_{\mathrm{CaS},\infty}(V_{\mathrm{th}}) h_{\mathrm{CaS},\infty}(V_{\mathrm{th}})\right) + \beta_{\mathrm{Ca}} \tag{2.18}$$

$$\frac{\partial m_{\mathrm{KCa},\infty}}{\partial V}(V, \mathrm{Ca})\Big|_{\mathrm{Ca}_\infty} = \frac{3}{(\mathrm{Ca}_\infty + 3)^2} f_1(V); \quad m_{\mathrm{KCa},\infty}(V, \mathrm{Ca})\big|_{\mathrm{Ca}_\infty} = \frac{\mathrm{Ca}_\infty}{\mathrm{Ca}_\infty + 3} f_2(V) \tag{2.19}$$

In practice, the method used by Fyon et al., 2024 to circumvent this issue involves using an approximate value of Ca_∞ when $\bar{g}_{\mathrm{CaS}}$ needs to be compensated — since its exact value is unknown *a priori* when solving the system. This approximation is obtained using Eq. (2.18), assuming $\bar{g}_{\mathrm{CaS}} \approx 10\,\mathrm{mS\,cm^{-2}}$.

Using an approximate value has the advantage of reducing the problem to solving a linear system. However, the disadvantage is that the solution obtained is only an approximation, meaning the target value of $\boldsymbol{g}_{\mathrm{DICs}}$ is not achieved exactly. This phenomenon is demonstrated in Figure 2.15, where the actual DIC values obtained after the generation step are displayed. There is variability in the DICs space, and the target value is not precisely reached. While the resulting population appears broadly degenerate, no quantitative assessment of the variability introduced by this approximation is available in the literature. One could reasonably suspect that in certain regions of the DICs space, this approximation might pose significant issues. In other words, we do not know how small variations in the DICs space impact the degenerate proximity and we assume that even small variations could lead to significant changes in neuronal activity. Consequently, the fact that the target value is not precisely reached is assumed problematic, as it may result in populations that are not functionally equivalent despite being *a priori* classified as degenerate. This work will introduce an alternative — **The Iterative Compensation Algorithm** — to the approximate method proposed by Fyon et al., 2024 in Section 5.2. This solution is presented as a major contribution of this Master Thesis.

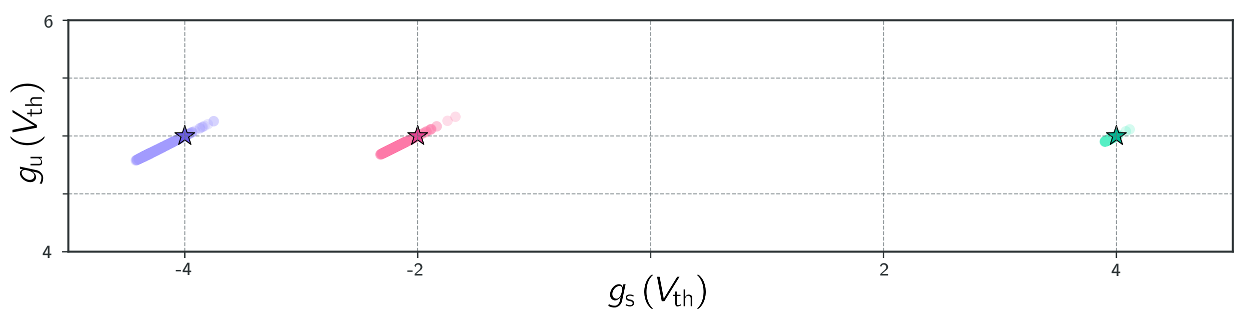


Figure 2.15: Comparison of target and obtained DIC values in the generated populations. The scatter plot represents the distribution of DIC values across populations of $M = 500$ instances. Each color corresponds to a different compensated population, with dots indicating individual instances and stars marking the target DIC values. The spread of points around the targets illustrates the variability introduced by the compensation process when following the approximation method from Fyon et al., 2024.

Finally, the choice of conductances to compensate is a non-trivial and important question. Indeed, a simple example demonstrates that, although not explicitly stated in the description of Algorithm 1, nothing inherently ensures that the compensated conductances remain within a biologically plausible range — that is, positive values. Figure 2.16 illustrates the results obtained in the STG model when compensating for $\bar{g}_{\mathrm{A}}$ and $\bar{g}_{\mathrm{CaS}}$ (Fig. 2.16A) while generating a degenerate population, but also when compensating for $\bar{g}_{\mathrm{H}}$ and $\bar{g}_{\mathrm{CaS}}$ (Fig. 2.16B), or $\bar{g}_{\mathrm{H}}$ and $\bar{g}_{\mathrm{CaT}}$ (Fig. 2.16C). Details are available in Appendix E. The first choice leads to negative conductance values, making the solution biophysically meaningless. In contrast, the second and third choices result in positive and thus valid solutions.

Lastly, Figure 2.17 highlights the importance of the variations required in the compensated conductances to reach a given point in the DICs space. This is demonstrated for both the case of compensating $\bar{g}_{\mathrm{H}}$ and $\bar{g}_{\mathrm{CaS}}$ and the case of compensating $\bar{g}_{\mathrm{H}}$ and $\bar{g}_{\mathrm{CaT}}$. Several observations can be made from this experiment, raising important questions.

The first observation is that not all compensations yield biophysically valid solutions within our model. In other words, the choice of compensated conductances affects the range of possible solutions. A natural question follows: is there a general rule or heuristic that allows us to identify, *a priori*, which conductance combinations will lead to biophysically valid solutions ? In other words, can we predict which conductances can be compensated without the risk of obtaining negative values ? This question relates to the concept of *reachability* of a point in the DICs space, depending on the set of conductances chosen for compensation. This work explores this question in Section 5.3 as a major contribution.

The second observation is that the choice of compensated channels influences the cost of compensation in terms of the required variation, as illustrated in Figure 2.17. From a control perspective, this raises questions about the costs associated with modulation and whether there exists an optimal way to modulate a population in practice, considering this cost. These questions could become highly significant in the future, as another recent study by Fyon et al., 2023 demonstrates that it is possible to construct a robust controller based on DICs theory and the compensation method.

For example, as shown in Figure 2.17, it is clear that the control cost is significantly higher when compensating $\bar{g}_{\mathrm{CaT}}$ instead of $\bar{g}_{\mathrm{CaS}}$, assuming that, biologically, the effective maximal conductance of these two channels varies at a similar cost per unit of change. A first approach to understanding these differences in cost lies in analyzing the different sensitivities, as presented in Figure 2.13. These questions regarding control costs are not explored further in this work and remain open. In our view, they represent one of many avenues for future research in computational neuroscience.

2.4.3 Contextualizing This Thesis

In conclusion, the method proposed by Fyon et al., 2024 is highly powerful due to its efficiency. With the additional contributions detailed in Sections 5.2 (on the iterative compensation algorithm) and 5.3 (on reachability), it enables extensive exploration of the DICs space and the generation of degenerate populations with precision. This will be particularly valuable for generating the large dataset required to train the deep learning tool at the core of this work.

The remaining gap in the current framework is that we do not know the DIC values associated with a given neuronal activity. While we can generate degenerate populations, there is no real quantitative mapping between the activity and the DICs.

Our thesis addresses this critical gap by developing a data-driven, deep learning pipeline that predicts the target DIC values associated with recorded neuronal activity. These predicted values can then be used as inputs for the generation method to obtain degenerate populations of CBMs with the desired target activity. The combination of the DICs theory, the efficient generation method, and our predictive pipeline enables a full mapping from neuronal recordings of spike times to a complete degenerate population of CBMs:

$$\underbrace{\text{Spike times record} \xrightarrow{\text{Inferring DIC values}} \boldsymbol{g}_{\mathrm{DICs}} \in \mathbb{R}^3}_{\texttt{The pipeline developed in this work}} \underbrace{\xrightarrow{\text{Generate a Degenerate Population}} \bar{\boldsymbol{g}} \in \mathbb{R}_+^{N_{\mathrm{model}}}}_{\substack{\texttt{Improvement of the existing method} \\ \texttt{through major contributions of this work} \\ \texttt{(iterative compensation algorithm and reachability)}}} \tag{2.20}$$

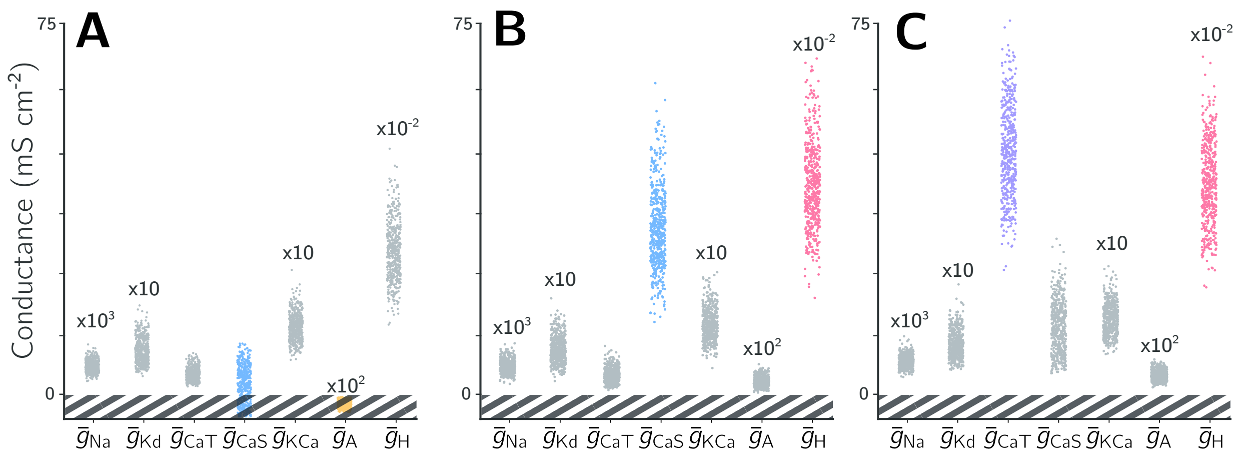


Figure 2.16: Impact of compensation choices on conductance validity in the STG model. Each scatter plot represents the distribution of maximal conductances across a population of $M = 500$ instances, with gray dots indicating non-compensated conductances and colored dots representing compensated ones. The values are displayed as scaled versions of the true conductances, with the appropriate scaling factor indicated above each column. The striped region corresponds to negative conductance values, which are biophysically meaningless. In all three cases, the populations are modulated to reach the same target DIC values but with different sets of compensated conductances. (A) Compensation of $\bar{g}_A$ and $\bar{g}_{CaS}$ results in some conductance values falling into the negative region, leading to invalid solutions. (B) Compensation of $\bar{g}_H$ and $\bar{g}_{CaS}$ avoids this issue, yielding a set of valid, positive conductances. (C) Similarly, compensating $\bar{g}_H$ and $\bar{g}_{CaT}$ maintains biophysical plausibility.

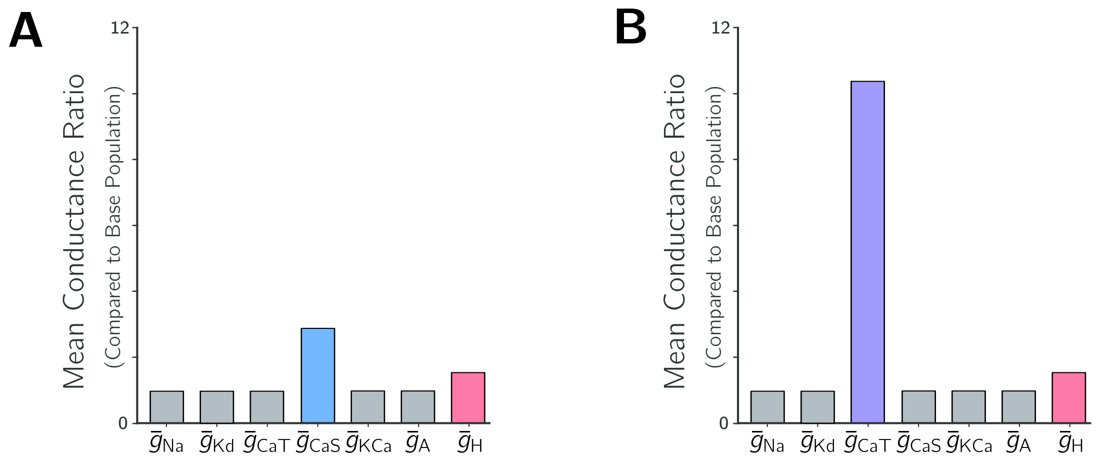


Figure 2.17: Mean conductance ratio for different compensation strategies in the STG model. The bar plots show the mean ratio of various maximal conductances over a population of $M = 500$ instances when two different sets of conductances are compensated. Gray bars represent conductances that are not compensated, while colored bars indicate the compensated conductances. (A) Compensation of $\bar{g}_{CaS}$ and $\bar{g}_H$ results in a moderate adjustment of both conductances. (B) Compensation of $\bar{g}_H$ and $\bar{g}_{CaT}$ leads to a similar adjustment in $\bar{g}_H$ but a compensation of $\bar{g}_{CaT}$ that is approximately 5 times larger than the $\bar{g}_{CaS}$ compensation in (A). This significant difference highlights how the choice of compensated conductances influences the magnitude of required adjustments, raising important questions about the cost and feasibility of different compensation strategies.

Box V – A Shortcut to Degenerate Population Generation

Efficient Algorithm for Generating Degenerate Populations

The efficient algorithm proposed by Fyon et al., 2024 provides a computationally efficient method for generating degenerate populations of conductance-based models (CBMs) by leveraging the Dynamic Input Conductances (DICs) theory. This approach significantly enhances the process of generating populations that exhibit desired spontaneous activity patterns, such as bursting or spiking, without the need for post-selection or filtering. The major features and contributions of this method are summarized below:

- **Direct degenerate population generation:** The algorithm generates populations that are *a priori* degenerate, ensuring that each individual within the population exhibits the desired spontaneous activity patterns. This eliminates the need for a trial-and-error approach, making the process computationally efficient.
- **Control over homogeneous scaling and ratio variability:** The method provides precise control over two fundamental mechanisms contributing to degeneracy: *homogeneous scaling* and *variability in conductance ratios*.
- **Two-step generation process:**
  1. **Step 1 - Spontaneous spiking initialization:** The algorithm begins by generating a population with spontaneous spiking behavior by enforcing a strongly negative value of g_{f}, which encourages the initiation of action potentials.
  2. **Step 2 - Refinement to desired DIC values:** The population is then refined to enforce specific DIC values (g_{s} and g_{u}), ensuring that the desired neuronal activity patterns are maintained.
- **Compensation algorithm:** The algorithm uses a compensation method derived from DICs theory to adjust specific conductances, ensuring that the resulting population exhibits the desired spontaneous activity. The compensation process involves solving a linear system based on the sensitivity matrix $\boldsymbol{S}$.

Highlighting the Reachability Question and the Iterative Compensation Needs

While the method itself offers a systematic approach to generating degenerate populations, this work introduces two fundamental questions that have not been explicitly addressed in the literature. The exploration of reachability and iterative compensation represents a major contribution of this thesis, highlighting important constraints and methodological considerations in the field.

- **Reachability and Control Costs:** This work introduces the concept of *reachability*, which explores the constraints on which combinations of conductances can be compensated to achieve a given point in the DICs space. Additionally, it raises questions about the *control costs* associated with different compensation strategies, highlighting the need for further investigation into optimal modulation techniques.
- **Iterative Compensation Algorithm:** To address the limitations of the original method, this work proposes an *iterative compensation algorithm* that improves the accuracy of targeting specific DIC values, especially in complex models like the STG where the sensitivity matrix depends on internal variables such as calcium concentration.

Application in This Work

In this thesis, this method will be used to efficiently generate populations of degenerate instances. It serves as the interface between the inference results from our pipeline and the final population of degenerate CBMs, enabling a full mapping from neuronal recordings of spike times to a complete degenerate population of CBMs.

$$\underbrace{\text{Spike times record} \xrightarrow{\text{Inferring DIC values}} \boldsymbol{g}_{\text{DICs}} \in \mathbb{R}^3}_{\texttt{The pipeline developed in this work}} \underbrace{\xrightarrow{\text{Generate a Degenerate Population}} \bar{\boldsymbol{g}} \in \mathbb{R}_+^{N_{\text{model}}}}_{\substack{\texttt{Improvement of the existing method}\\ \texttt{through major contributions of this work}}}$$

Chapter 3

Background — Supervised Learning and Deep Learning

To effectively model and design complex data-driven systems, such as the pipeline developed in this thesis, it is essential to bridge theoretical principles with practical implementations. This chapter provides the necessary background in Supervised Learning (SL) and Deep Learning (DL), introducing the vocabulary and concepts required to understand the methods and models employed in this work.

Structure of the Chapter

The first section, **3.1 Supervised Learning**, introduces the basic terminology and mathematical framework used to explore patterns automatically by leveraging a finite set of example data points. It provides a rigorous formulation of the main objective of this thesis within such a framework: how can we extract a mapping from a spike train to the DICs space? The key points are summarized in Box VI.

The second section, **3.2 Deep Learning**, presents the specific building blocks that we will use throughout the thesis to construct the final mapping. It also discusses some of the main challenges associated with using DL. Box VII highlights the main takeaways from this section.

By the end of this chapter, the reader should have a solid understanding of the basic building blocks and their mathematical formulations, enabling to comprehend and replicate the work presented in the subsequent chapters.

3.1 Supervised Learning

3.1.1 The General Framework of Supervised Learning

This work falls within the scope of **supervised learning** (SL). The goal is to build a model from a **labeled dataset** (*supervised*) so that it can **generalize** (*learning*), i.e., perform well on previously unseen data. Deep Learning (DL) offers a powerful framework for constructing supervised learning (SL) models, and it serves as the primary approach explored in this thesis. This section introduces a general mathematical framework that defines key concepts in SL, such as the *labeled dataset*, the *model*, and *generalization*. These fundamental concepts also form the basis of DL.[1]

Learning a mapping from input to output The ultimate goal of supervised learning is to establish a mapping between observations, referred to as *inputs*, and a supervised signal, referred to as *outputs*. In this work, we aim to construct a function that maps a sequence of spike times as input to corresponding DIC values as output. In other words, we seek a function approximator with respect to labeled input and output.

[1]Once again, to avoid overloading the text, a single reference is made here (Bishop, 2006). This book serves as the primary source of information used for writing this section on supervised learning, particularly the introductory chapter. For both the supervised learning and deep learning sections, the following reference (Goodfellow et al., 2016) has also been extensively consulted, especially Chapters 3 ('*Probability and Information Theory*'), 4 ('*Numerical Computation*'), 5 ('*Machine Learning Basics*'), and 11 ('*Practical Methodology*'). Finally, the full class of Louppe, 2023 has also been an important reference.

Supervised learning assumes the existence of an unknown function f that maps the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. This function may be deterministic or stochastic. The objective is to approximate f with a parametric function f_θ, i.e., a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ such that $f_\theta \approx f$. To achieve this, SL seeks optimal values for the parameters $\theta \in \Theta$, leveraging a finite labeled dataset $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{N}$. This dataset results from independent and identically distributed (i.i.d.) samples $(x_i, y_i) \sim p_{\mathcal{X},\mathcal{Y}}$. The probabilistic framework facilitates formalizing uncertainty (measurement noise, inherent stochasticity of the problem, etc.), and the goal is to construct an estimator of the conditional distribution $p_{\mathcal{Y}|\mathcal{X}}$.

Supervised learning methods address several fundamental questions:

- *What defines a good parameter set?*
- *How to find the best or at least good values for $\theta \in \Theta$?*
- *How to choose the structure of the model function f_θ?*

The **loss function** quantifies model quality: $f_\theta \mapsto L[f_\theta]$. Typically, the loss function provides a scalar measure of model performance that we aim to minimize. Thus, the learning algorithm solves an optimization problem:

$$\underbrace{f_{\theta^*} = \arg\min_{f_\theta \in \mathcal{F}} L[f_\theta]}_{\texttt{Function-space (meta)-learning}} \quad \Leftrightarrow \quad \underbrace{\theta^* = \arg\min_{\theta \in \Theta} L[f_\theta]}_{\texttt{Parameter-space learning}} \tag{3.1}$$

During training, the right-hand formulation is more explicit as it emphasizes that the learnable parameter space Θ is explored. The loss function measures the similarity between model predictions $f_\theta(x_i)$ and true outputs y_i. Lower loss implies higher similarity, and the associated term '*risk*' represents the loss value, which we seek to minimize.

The **learning algorithm** determines how the parameter space is explored based on the loss values — in other words, it defines how we solve the minimization problem.

The choice of the structure of the model function f_θ, along with the parameterization of learning algorithms, concerns *meta-learning*. In this context, *hyperparameters* refers to tunable quantities that are not directly optimized by the *main* learning algorithm. These can span a wide range, including the design of the model architecture, the choice and configuration of the learning algorithm, and other components that shape the learning process. We denote hyperparameters as $\psi \in \Psi$, while *learnable parameters* — those updated during the main learning — are denoted as $\theta \in \Theta$. As hyperparameters are selected *a priori*, they influence the structure of $\mathcal{F}$ or the way it is explored, thereby linking them to the left-hand side of the optimization formulation.

Learning is being able to generalize A model is considered effective when it can **generalize**, meaning it not only learns the patterns in the training set but also makes accurate predictions on unseen data. Generalization is translated into learning through two principles:

- **Minimization of the expected risk** A model that generalizes well is close to the true distribution $p_{\mathcal{X},\mathcal{Y}}$, allowing the loss function minimization to be expressed as:

$$f_{\theta^*} = \arg\min_{f_\theta \in \mathcal{F}} L[f_\theta] \overset{\textbf{Expected risk minimization}}{=} \arg\min_{f_\theta \in \mathcal{F}} \mathbb{E}_{(x,y)\sim p_{\mathcal{X},\mathcal{Y}}} \left[\mathcal{L}(f_\theta(x), y)\right]. \tag{3.2}$$

  In practice, $L[f_\theta]$ cannot be directly evaluated since computing the expectation over the joint probability requires knowing it. The function $\mathcal{L}$ differs from L in that it takes a random variable as input and returns another random variable, whereas L outputs a summary scalar.

- **Minimization of the empirical risk** Since $p_{\mathcal{X},\mathcal{Y}}$ is unknown, we estimate the expectation using the dataset $\mathcal{T}$ and apply the empirical risk minimization principle:

$$f_{\theta^*} = \arg\min_{f_\theta \in \mathcal{F}} L[f_\theta] \overset{\textbf{Empirical risk minimization}}{\approx} \arg\min_{f_\theta \in \mathcal{F}} \frac{1}{|\mathcal{T}|} \sum_{(x_i,y_i)|_{i=1}^{N} \in \mathcal{T}} \ell\left(f_\theta(x_i), y_i\right). \tag{3.3}$$

  The function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ measures similarity (low values indicate high similarity). This transition from expected risk minimization to empirical risk minimization introduces challenges such as **overfitting**.

We emphasize that it is empirical risks that are used in practice, even though we are interested in expected risks. Thus *overfitting* is invariably at the center of discussions in SL.

Overfitting and dataset splitting Overfitting occurs when a model achieves a low empirical risk on the training data but fails to generalize well to new, unseen data. This phenomenon contradicts the principle of expected risk minimization and arises due to the transition from expected risk to empirical risk. Overfitting is caused by a model that adapts too closely to the specific characteristics of the training sample, capturing noise rather than underlying relevant patterns. Detecting and mitigating overfitting is crucial in supervised learning.

To detect overfitting, the dataset must be divided into distinct subsets:

$$\mathcal{T} \longrightarrow \{\mathcal{T}_{\text{train}}, \mathcal{T}_{\text{val}}, \mathcal{T}_{\text{test}}\},$$

- **Training set** $\mathcal{T}_{\text{train}}$: Used to adjust the model parameters during the training phase.
- **Validation set** $\mathcal{T}_{\text{val}}$: Used to fine-tune hyperparameters and evaluate model performance on data not seen during training for a specific set of hyperparameters.
- **Test set** $\mathcal{T}_{\text{test}}$: Used exclusively after final training to estimate the model actual ability to generalize to new data.

This data splitting is fundamental to get correct performance estimates of the model (Kaufman et al., 2012) and is considered one of the crucial elements in this work. Special attention is given to avoiding *data leakage* in all experiments conducted in this Master Thesis. **Data leakage** refers to the use of information from one dataset split in a phase where the model is not supposed to have access to it, e.g. training using the validation set and then using the performance on it to choose the best set of hyperparameters.

Overfitting can be identified by a significant performance difference between different dataset splits:

$$\frac{1}{N_{\text{train}}} \sum_{(x_i,y_i) \in \mathcal{T}_{\text{train}}} \ell\left(f_\theta(x_i), y_i\right) \ll \frac{1}{N_{\text{test}}} \sum_{(x_i,y_i) \in \mathcal{T}_{\text{test}}} \ell\left(f_\theta(x_i), y_i\right) \implies \texttt{overfitting}$$

Thus, during training phases, both the *training loss* and the *validation loss* will be tracked. Performance on the test set will only be evaluated at the very end of the experiments to prevent any *leakage* influencing personal decision-making. Indeed, it is common to observe unintended leakage from experimenters who review test set results and are subconsciously influenced by them, even though no direct automatic criterion is assessed on these results. As a safeguard, the test set is not generated until the very end of this work.

Among the existing methods to mitigate overfitting, data augmentation and regularization are employed in this work. These methods will be introduced in detail in the following.

3.1.2 A Rigorous Formulation of the Specific Supervised Learning Problem in This Work

This section provides a rigorous formulation of the supervised learning problem addressed in this Master Thesis. The objective is to learn a mapping between a sequence of spike times and the space of DIC values. In this context, we formally define the structure of the input $\boldsymbol{x}_i \in \mathcal{X}$ and the output $\boldsymbol{y}_i \in \mathcal{Y}$, which are collected in the dataset $\mathcal{T} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$.

The input is represented as a vector of spike times:

$$\boldsymbol{x} = [t_1, t_2, \ldots, t_{L_x}] \in \mathbb{R}_+^{L_x \times 1}. \tag{3.4}$$

Here, L_x denotes the length of the sequence, which varies from one neuronal activity to another. Neurons exhibiting more spikes (e.g., during bursting events) have a greater number of spike times within a fixed period of time compared to neurons with little to no activity. The $\times 1$ notation emphasizes that only a single quantity (time) is recorded for each detected spike.

The output is represented by the vector of DIC values:

$$\boldsymbol{y} = [g_\text{s}, g_\text{u}] \in \mathbb{R}^2, \tag{3.5}$$

which corresponds to the DIC values at the threshold potential. As introduced in Section 2.4, the exact value of fast conductance is of no interest, provided it is sufficiently negative. Thus, the main interest lies in the slow and ultra-slow conductances. The latter are the necessary input for the generation algorithm. In other words, the pipeline aims to control the second generation step — modulated through g_s and g_u (Algorithm 1).

Since the sequence length varies from one instance to another, we can express the learning objective in a general form as using the dataset $\mathcal{T}$ to learn a function:

$$f_\theta : \mathbb{R}_+^{* \times 1} \to \mathbb{R}^2, \tag{3.6}$$

where the notation $* \times 1$ represents the dimensions of a matrix with an arbitrary and variable number of rows and a single column.

Box VI – A Framework for Supervised Learning

The General Framework of Supervised Learning

Supervised learning (SL) aims to approximate an unknown function f that maps inputs from a domain $\mathcal{X}$ to outputs in a domain $\mathcal{Y}$. The goal is to construct a parametric function $f_\theta : \mathcal{X} \to \mathcal{Y}$ such that $f_\theta \approx f$, leveraging a finite labeled dataset $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{N}$. The learning process involves minimizing a loss function $L[f_\theta]$ to optimize the parameters θ, with the ultimate aim of generalizing well to unseen data. The framework emphasizes minimizing the empirical risk, being approximated using the training dataset, with the true objective of minimizing the expected risk.

Part of This Thesis as a SL Problem

This thesis addresses a specific supervised learning problem: learning a mapping between sequences of spike times and the space of DIC values. The input is represented as a vector of spike times, while the output is a vector of DIC values. The variable-length nature of the input sequences introduces challenges in designing a suitable architecture. The learning objective is formalized as:

$$f_\theta : \mathbb{R}_+^{* \times 1} \to \mathbb{R}^2, \boldsymbol{x} \mapsto \boldsymbol{y}$$

where the notation $* \times 1$ denotes a matrix with an arbitrary number of rows and a single column. This formulation encapsulates the essence of the supervised learning task in this work.

3.2 Deep Learning

The tools used in the supervised learning methods of this work are derived from the theory of *Deep Learning* (DL).

Deep learning encompasses a set of building blocks that can be assembled to tackle supervised learning problems — as well as unsupervised problems, which are largely outside the scope of this work — in a generic and highly efficient manner. In recent years, deep learning has become central to a technological, technical, and social revolution. The theoretical foundations of it have been known and explored since the 1980s. As evidence of this revolution, John J. Hopfield and Geoffrey Hinton were awarded the Nobel Prize in Physics in 2024 for their groundbreaking work on neural networks, which form the core of deep learning (Nobel Prize Outreach, 2024).

This section aims to present the specific aspects of deep learning used in this work. This includes the various building blocks, insights about them, and training methods.

Setup of the learning process in supervised learning The full training process — from raw data to the final trained pipeline — consists of several essential steps to ensure effective learning. These steps aim to balance model accuracy, generalization, and computational efficiency. Below is a detailed breakdown of the major components of setting up the learning process in supervised learning, specifically when using DL:

1. **Choice of features:** The first step in supervised learning is determining which features (or inputs) from the data will be used by the model. In general SL applications, this step may involve selecting a subset of available data. However, in this work, the task is to predict DICs from a spike train, and the spike train serves as the sole input available at the start of the pipeline. Therefore, this feature selection step takes the form of a transformation of the spike times sequence as well as its augmentation.

   The transformation step, often referred to as *feature engineering*, involves manipulating the spike train to generate a set of features that are usable by the learning algorithm. Numerous methods exist for transforming a spike train into features, and these depend heavily on the architecture used. A straightforward approach, applicable to most SL algorithms, would be to extract summary statistics from the spike train, such as activity type, firing rate, bursting frequency, etc. This approach is used by Prinz et al., 2003, where the inputs to the database include such summary statistics. In contrast, this work opts to retain the complete sequence of spike times as the primary input to the pipeline. This choice and its constraints will be discussed in more detail in the following of this section.

   $$(x_i, y_i) \in \mathcal{T} \xrightarrow{\texttt{Data Transformation (Feature Engineering)}} (x_i', y_i) \in \mathcal{T}', \quad \text{with } x_i \in \mathcal{X},\ x_i' \in \mathcal{X}'.$$

   The augmentation step, on the other hand, takes place during the training phase, i.e., when exploring the parameter space. Thus, during inference — i.e., when the model is used for real-world predictions — no augmentation occurs. The purpose of data augmentation is to transform a dataset $\mathcal{T}$ into a similar dataset of larger size:

   $$\mathcal{T} \xrightarrow{\texttt{Data Augmentation}} \mathcal{T}', \quad \text{with } |\mathcal{T}| < |\mathcal{T}'|.$$

   In this work, this augmentation will typically be achieved by introducing noise into the spike times sequence or by using only a part of the sequence.

2. **Choice of architecture and its hyperparameters:** The next step involves selecting the architecture of the function f_θ, which will process the inputs into outputs and whose internal state will be optimized by the learning algorithm. In this work, deep learning is used as the framework for this architecture. It can be described by several hyperparameters, such as the number of blocks, the dimensions of

specific vectors, and so on, which are not learned by the training algorithm but can significantly impact the results of the model.

3. **Choice of loss function:** There are many loss functions available in supervised learning. In deep learning, we focus on a broad set of loss functions with the constraint that the measure must be differentiable with respect to the learnable parameters.

4. **Choice of learning algorithm:** This refers to the method used to explore the space of learnable parameters. It is the procedure responsible for updating the parameters $\theta \in \Theta$. In deep learning, various types of learning algorithms exist, but they almost all rely on the principle of *gradient descent*. The choice of learning algorithm dictates how the model parameters are adjusted to minimize the chosen loss function.

5. **Tuning hyperparameters:** After selecting the architecture and the learning algorithm, the hyperparameters of the model need to be tuned. These include parameters such as the learning rate, batch size, number of epochs, and other architectural aspects. Hyperparameter tuning is crucial for finding the optimal configuration of the model, and this process is typically carried out using techniques such as grid search, random search, or more advanced methods like Bayesian optimization. In this work, we use *random search*, as detailed by Bergstra and Bengio, 2012 and in Section 3.2.3.

6. **Final training:** Once the architecture, loss function, learning algorithm, and hyperparameters are chosen and tuned, the model undergoes final training. The goal of this phase is to optimize the model parameters θ based on the entire training dataset, achieving the best possible performance.

7. **Evaluation on test set:** Finally, after training, the model is evaluated on the test set — a dataset that has never been seen before by the model. This evaluation gives an estimate of the model ability to generalize to new, unseen data. Performance on the test set is used as the final measure of model success, and this step is crucial in ensuring that the model has not overfitted to the training or validation data.

3.2.1 Representation Learning Through Highly Parametrized Layers

An Encoder-Decoder Paradigm

Deep learning introduces the function $f_\theta : \mathcal{X} \to \mathcal{Y}$ in the form of a deep neural network (DNN). This DNN architecture is characterized by a succession of transformations, each corresponding to a *layer*. The term *deep* neural network comes from the fact that multiple layers are stacked:

$$\boldsymbol{y} = f_\theta(\boldsymbol{x}) \xrightarrow{\texttt{Deep Neural Network (DNN)}} \boldsymbol{y} = \left(f_{L,\theta_L} \circ f_{L-1,\theta_{L-1}} \circ \cdots \circ f_{0,\theta_0}\right)(\boldsymbol{x}).$$

Each layer is parametrized, and these parameters are explored by the learning algorithm. The power of DNNs lies in their high representation capacity, meaning that by appropriately learning the parameters θ, they can approximate a wide range of functions.

One interpretation of the successive transformations applied by a DNN is through the concept of *representation learning*. Each layer of the architecture projects the input space into a new space (possibly of different dimension), and the learning algorithm successively learns different representations of the data. The full architecture thus learns how to transform the input representation into its corresponding output representation. In this work, we use an architecture that can be visualized as two distinct parts: an *encoder* and a *regression head* (decoder).

The encoder learns to represent the sequence of spike times, of variable length, in a fixed-dimensional space. Handling features of variable length poses a challenge because a lot of deep learning building blocks are *structurally* unable to manage variable size. A commonly used alternative is to rely on summary statistics. However, certain deep learning architectures are designed to process sequences of variable length. Among the most commonly used architectures are *recurrent neural networks* (Petneházi, 2019), *convolutional neural networks* (Borovykh et al., 2018), and finally, the architecture chosen in this work, *transformers (attention-based)* (Wen et al., 2023). The encoder $\mathcal{E} : \mathbb{R}^{*\times 1} \to \mathbb{R}^{d_{\text{latent}}}$ transforms a variable-length representation of neuronal activity, given as the sequence of spike times $\boldsymbol{x}$, into a representation in what is called a *latent space* — that is, a representation $\boldsymbol{z}_{\text{latent}}$ that is assumed to be meaningful but cannot be directly measured in the physical world. The latent space has a fixed, predefined dimension d_{latent}:

$$\textit{Encoding into the latent space: } \boldsymbol{z}_{\text{latent}} = \mathcal{E}(\boldsymbol{x}). \tag{3.7}$$

We justify the choice of an architecture that learns the encoding, rather than relying on a model based on summary statistics. The latter approach being essentially equivalent to forcing the latent space to be the space of selected statistics. Our argument in favor of learned encoding is that, in this context, we should not impose a prior choice on the information to be extracted from the sequence of spike times. With this architecture, the model learns to extract the features that are most relevant for minimizing the loss function, i.e., for predicting the DIC values.

However, we emphasize that this choice comes at a cost. The encoder introduces a significant number of additional parameters to be learned, and it should be known that this part of deep learning architectures substantially increases both the data requirements and the computational cost of training.

Finally, we use an *attention-based* encoder as it naturally handles variable-length spike times sequences and captures long-range dependencies without sequential processing. Unlike CNNs, which rely on fixed local receptive fields, or RNNs, which process data step by step, making training slower, self-attention dynamically weights spike times. However, the overall approach remains flexible, and replacing the encoder with a CNN or RNN would still be feasible without major modifications.

The regression head learns the mapping between the latent representation and the space of DIC values. Once we have a fixed-dimensional latent representation, we enter a more standard deep learning context, where the goal is to transform this representation into another fixed-dimensional space — namely, the output space. We define $\mathcal{R}_{\text{DICs}} : \mathbb{R}^{d_{\text{latent}}} \to \mathbb{R}^2$ as the part of the architecture responsible for decoding the latent representation into the DICs space:

$$\textit{Decoding into the DICs space: } \boldsymbol{y} = \mathcal{R}_{\text{DICs}}(\boldsymbol{z}_{\text{latent}}). \tag{3.8}$$

In summary, the architecture follows an encoder-decoder paradigm, where the goal is to encode a variable-length input into a fixed-dimensional latent space — or in other words, extract meaningful summary features from variable-length inputs — and then decode it into a fixed-dimensional output space — that is, transform the summary features into a useful representation:

$$\underbrace{\boldsymbol{x} \in \mathbb{R}^{*\times 1} \xrightarrow{\texttt{Encoder } \mathcal{E}} \boldsymbol{z}_{\text{latent}}}_{\text{Projection into latent space}} \in \underbrace{\mathbb{R}^{d_{\text{latent}}} \xrightarrow{\texttt{Decoder } \mathcal{R}_{\text{DICs}}} \boldsymbol{y} \in \mathbb{R}^2}_{\text{Mapping to DICs space}}. \tag{3.9}$$

3.2.2 The Building Blocks of Deep Learning Architectures

This section presents the various building blocks used in this work. In a typical scenario, each building block can be thought of, perceived, and described as an input-output transformation. Thus, the layer-wise description of DNNs translates into a sequence of transformations where the output of one block becomes the input of the next, from the input of the pipeline to its output:

$$\boldsymbol{x} \to \boldsymbol{z}_1 = f_{0,\theta_0}(\boldsymbol{x}) \to \boldsymbol{z}_1 = f_{1,\theta_1}(\boldsymbol{z}_1) \to \cdots \to \boldsymbol{z}_{L-1} = f_{L-1,\theta_{L-1}}(\boldsymbol{z}_{L-2}) \to \boldsymbol{y} = f_{L,\theta_L}(\boldsymbol{z}_{L-1}) \Leftrightarrow \boldsymbol{y} = f_\theta(\boldsymbol{x}),$$

where each building block is a fundamental parametric construction in the sense that it can be described independently from the others.

In the following, $\boldsymbol{x}$ and $\boldsymbol{z}$ always denote the input and the output of the layer, respectively. The Figure 3.1 illustrates each layer and allows the reader unfamiliar with deep learning concepts to get an idea of the function of each of them.

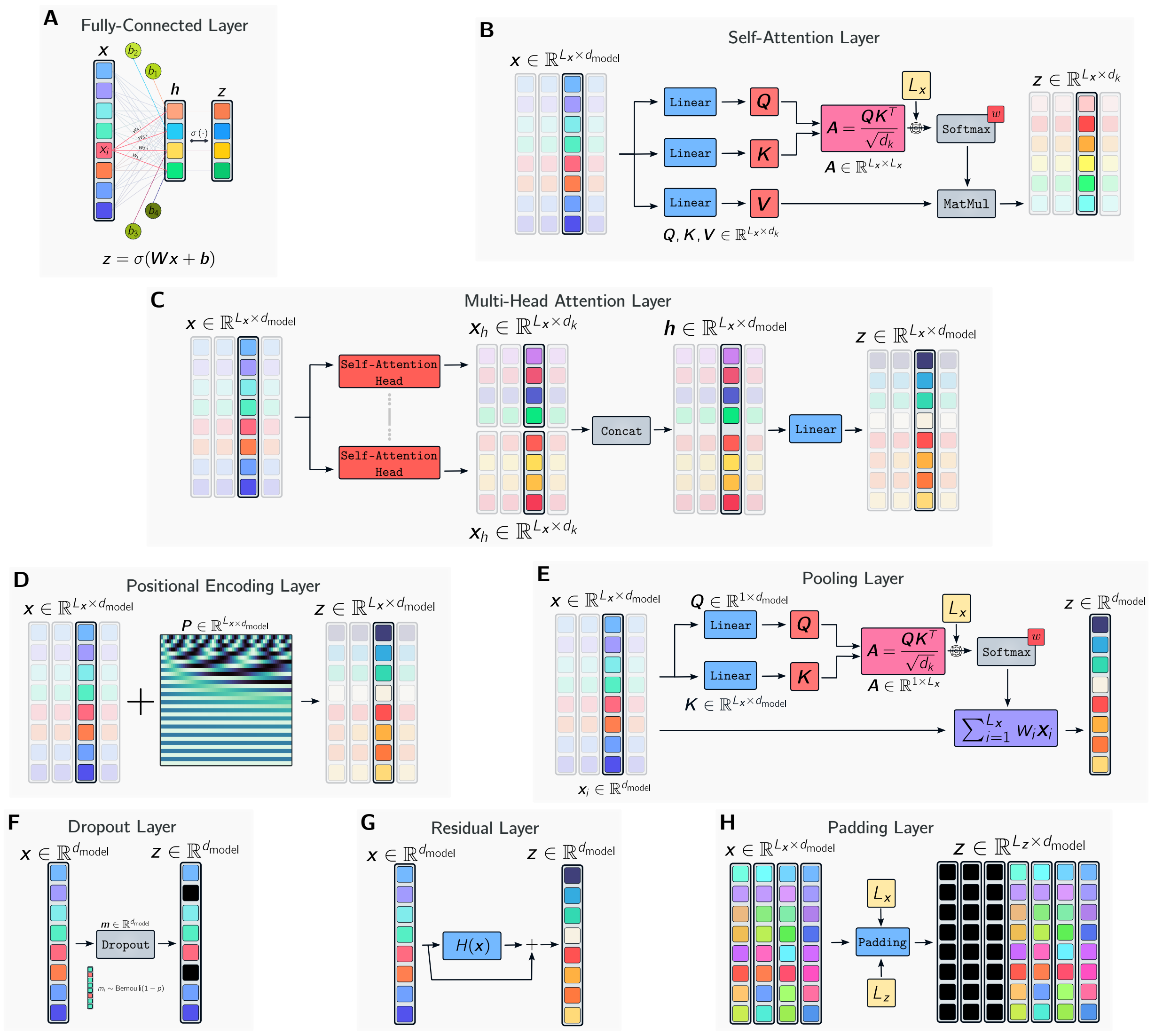

A
Fully-Connected Layer
$z = \sigma(Wx + b)$
B
Self-Attention Layer
$x \in \mathbb{R}^{L_x \times d_{\text{model}}}$
Linear
Softmax
MatMul
$A = \frac{QK^T}{\sqrt{d_k}}$
$A \in \mathbb{R}^{L_x \times L_x}$
$Q, K, V \in \mathbb{R}^{L_x \times d_k}$
$z \in \mathbb{R}^{L_x \times d_k}$
C
Multi-Head Attention Layer
Self-Attention Head
Concat
Linear
$x_h \in \mathbb{R}^{L_x \times d_k}$
$h \in \mathbb{R}^{L_x \times d_{\text{model}}}$
$z \in \mathbb{R}^{L_x \times d_{\text{model}}}$
D
Positional Encoding Layer
$P \in \mathbb{R}^{L_x \times d_{\text{model}}}$
E
Pooling Layer
$Q \in \mathbb{R}^{1 \times d_{\text{model}}}$
$K \in \mathbb{R}^{L_x \times d_{\text{model}}}$
$A \in \mathbb{R}^{1 \times L_x}$
$\sum_{i=1}^{L_x} w_i x_i$
$x_i \in \mathbb{R}^{d_{\text{model}}}$
$z \in \mathbb{R}^{d_{\text{model}}}$
F
Dropout Layer
Dropout
$m \in \mathbb{R}^{d_{\text{model}}}$
$m_i \sim \text{Bernoulli}(1-p)$
G
Residual Layer
$H(x)$
H
Padding Layer
Padding
L_x
L_z
$z \in \mathbb{R}^{L_z \times d_{\text{model}}}$

Figure 3.1: Building blocks of deep learning architectures. This figure illustrates the fundamental components used in constructing deep learning models, each depicted as an input-output transformation. (A) The fully-connected layer performs a linear transformation followed by a non-linear activation function. (B) The self-attention layer computes a weighted sum of values based on query-key similarity, enabling dynamic focus on relevant parts of the input sequence. (C) The multi-head attention layer extends self-attention by computing attention in parallel across multiple heads, enhancing expressiveness. (D) The positional encoding layer adds positional information to input sequences, allowing the model to understand the order and relative positions of elements. (E) The pooling layer aggregates information to reduce dimensionality, transforming variable-length inputs into fixed-size representations. (F) The dropout layer randomly sets a fraction of input units to zero during training to prevent overfitting. (G) The residual layer provides direct pathways for information transfer, helping to mitigate the vanishing gradient problem. (H) The padding layer adjusts the length of shorter sequences by adding padding elements, ensuring uniform sequence lengths within a batch. These building blocks are essential for constructing robust and efficient deep learning architectures.

The fully-connected layer This is the most basic and classical layer in deep learning (Fig. 3.1A). A fully-connected (FC) layer can be described by the following transformation:

$$\boldsymbol{z} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{3.10}$$

where $\boldsymbol{W}$ represents a weight matrix, $\boldsymbol{b}$ is a bias vector, and σ is a non-linear activation function applied element-wise, typically chosen to introduce non-linearity into the model (e.g., ReLU[2], Sigmoid, or Tanh). The purpose of the FC layer is to learn the relationship between the input features and the output through these parameters $\boldsymbol{W}$ and $\boldsymbol{b}$, which are the learnable parameters.

In this layer, every element of the input vector $\boldsymbol{x}$ is connected to every element of the output vector $\boldsymbol{z}$, making it a dense layer. The transformation is a linear combination of the input values, followed by a non-linear activation. The non-linearity allows the network to model complex relationships between the input and output. When we refer to a *linear* layer, we mean a fully-connected layer without any non-linear activation applied.

In the case of sequence data, such layers are typically applied point-wise, that is, independently at each position of the sequence.

Self-attention layers Self-attention mechanisms are a core component of models like the Transformer (Vaswani et al., 2023), allowing them to dynamically focus on different parts of the input sequence based on contextual relevance. The self-attention operation computes a weighted sum of values, where the weights are determined by the similarity between queries and keys (Fig. 3.1B). Given an input sequence represented by a matrix $\boldsymbol{x} \in \mathbb{R}^{L_{\boldsymbol{x}} \times d_{\text{model}}}$ (where $L_{\boldsymbol{x}}$ is the sequence length and d_{model} is the model dimensionality), the self-attention layer is defined as:

$$\boldsymbol{z} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}, \quad \boldsymbol{Q} = \boldsymbol{x}\boldsymbol{W}_Q,\ \boldsymbol{K} = \boldsymbol{x}\boldsymbol{W}_K,\ \boldsymbol{V} = \boldsymbol{x}\boldsymbol{W}_V \tag{3.11}$$

where $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are learned projection matrices shared across all time steps. The resulting matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{L_{\boldsymbol{x}} \times d_k}$ scale with the sequence length $L_{\boldsymbol{x}}$ — allowing the model to handle inputs of arbitrary length. Importantly, while the *shape of the outputs* depends on the sequence length, the *parameter matrices* are independent of it.

The attention weights are computed by taking the scaled dot product of $\boldsymbol{Q}$ and $\boldsymbol{K}^T$, yielding an attention score matrix in $\mathbb{R}^{L_{\boldsymbol{x}} \times L_{\boldsymbol{x}}}$. These scores are scaled by $\sqrt{d_k}$ and passed through a softmax to produce normalized attention weights. These weights are used to compute a weighted sum over the values $\boldsymbol{V}$, resulting in the final output $\boldsymbol{z} \in \mathbb{R}^{L_{\boldsymbol{x}} \times d_k}$, which contains the self-attended representations of the input.

To enhance expressiveness, attention is computed in parallel across multiple heads, forming the multi-head attention mechanism (Fig. 3.1C):

$$\boldsymbol{z} = \text{concat}\,(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_H)\,\boldsymbol{W}_O \tag{3.12}$$

where each $\boldsymbol{x}_h \in \mathbb{R}^{L_{\boldsymbol{x}} \times d_k}$ is the output of the h-th attention head, and $\boldsymbol{W}_O \in \mathbb{R}^{H \cdot d_k \times d_{\text{model}}}$ projects the concatenated result back into the model dimension.

In this work, we use H attention heads and set the dimensionality of each head to $d_k = d_{\text{model}}/H$, so that the output dimension matches the input. Thanks to this formulation, our model can process input sequences of variable lengths: the weight matrices are shared across all positions, and the self-attention computations naturally adapt to the length of the input by producing appropriately sized outputs.

[2]The name ReLU, short for Rectified Linear Unit, is a classic term in deep learning referring to the function applied element-wise $\boldsymbol{z} = \text{ReLU}(\boldsymbol{x}) = \max(0, \boldsymbol{x})$. In this work, we also explore GELU and SiLU that are ReLU-like functions (Hendrycks and Gimpel, 2023).

Positional encoding layers An inherent problem with the attention mechanism is its inability to distinguish the position of elements within a sequence. Therefore, it is common to introduce an explicit positional encoding mechanism into the sequence (Fig. 3.1D). Positional encoding allows the model to understand the order and relative positions of the input elements.

Mathematically, positional encoding can be defined as a function that adds positional information to the input embeddings. Given an input sequence $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{L_x}]$, the positional encoding $\boldsymbol{P}(i)$ is added to each element of the sequence:

$$z_i = \boldsymbol{x}_i + \boldsymbol{P}(i), \tag{3.13}$$

where $\boldsymbol{P}(i)$ is the positional encoding for the i-th position in the sequence.

A common approach to positional encoding is to use sine and cosine functions of different frequencies. For a given position i and dimension j, the positional encoding can be defined as:

$$P(i, 2j) = \sin\left(\frac{i}{10000^{2j/d_{\text{model}}}}\right), \tag{3.14}$$

$$P(i, 2j+1) = \cos\left(\frac{i}{10000^{2j/d_{\text{model}}}}\right), \tag{3.15}$$

where d_{model} is the dimensionality of the positional encoding. This encoding scheme allows the model to capture both the absolute and relative positions of the elements in the sequence.

By incorporating positional encoding through addition, the self-attention mechanism can effectively utilize the order of the sequence, enhancing its ability to model sequential data accurately. This approach is widely used in transformer-based architectures due to its simplicity and effectiveness.

Pooling layers Pooling layers reduce the dimensionality of feature representations by aggregating information, helping to manage variable-length inputs and improve computational efficiency. The use of a pooling layer is essential in this work to bridge the encoder and the decoder. It is through this layer that we transform a variable-length representation into a fixed-size representation, independent of the input length. Mathematically, pooling is characterized by a dimensionality reduction:

$$\boldsymbol{z} = \kappa(\boldsymbol{x}); \quad \boldsymbol{x} \in \mathbb{R}^{*\times d_x}, \quad \boldsymbol{z} \in \mathbb{R}^{d_z} \tag{3.16}$$

There are several possible forms for the function κ, such as taking the mean of the features over the sequence, the maximum of the features over the sequence, or, as we use in this work, *self-attention pooling* (Fig. 3.1E). This means that the aggregation is performed through a weighted average of the sequence elements, where the weights are functions of the sequence itself.

Normalization layers A phenomenon known as *internal covariate shift* refers to the variation in the distribution of inputs due to the updates of parameters. This is a problem because it slows down learning and exacerbates the vanishing gradient problem, as the distributions gradually shift toward the saturated regions of the activation function (Ioffe and Szegedy, 2015). A solution to this covariate shift is to use normalization layers, whose main function is to reduce the excessive deformation of internal distributions. Typically, normalization can be described by:

$$\boldsymbol{z} = \boldsymbol{\gamma}\frac{\boldsymbol{x} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})} + \boldsymbol{\beta} \tag{3.17}$$

where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameters, and μ and σ are statistics of the input, with the exact form depending on the type of normalization. In this work, we use *Layer Normalization* (Ba et al., 2016), which normalizes across the feature and sequence dimensions — meaning that each input sequence is normalized independently of the others.

Residual layers Residual layers, also known as *skip connections*, provide direct pathways for information transfer from a shallower layer to a deeper layer (Fig. 3.1G). Mathematically, a skip connection around a block $H(\boldsymbol{x})$ is expressed as:

$$\boldsymbol{z} = H(\boldsymbol{x}) + \boldsymbol{x} \tag{3.18}$$

There are two main reasons for using residual layers (He et al., 2016):

1. They help mitigate the vanishing gradient problem by providing direct gradient flow paths to the shallower layers.

2. They simplify the learning of an identity mapping ($H(\boldsymbol{x}) = \mathbf{0}$), which would otherwise be more difficult to learn in a standard FC layer. The benefit of the identity mapping is that it allows the network to avoid using certain blocks that introduce transformations too complex for the model.

Dropout Layers Dropout is a regularization technique used to prevent overfitting in neural networks (Hinton et al., 2012). During training, dropout randomly sets a fraction of the input units to zero at each update (Fig. 3.1F). This forces the network to learn redundant representations and reduces the risk of relying too heavily on any single feature. Mathematically, dropout can be described by:

$$\boldsymbol{m} \sim \text{Bernoulli}(1 - p_{\text{dropout}}), \quad \boldsymbol{z} = \boldsymbol{m} \odot \boldsymbol{x}, \tag{3.19}$$

where $\boldsymbol{m}$ is a binary mask drawn from a Bernoulli distribution, p_{dropout} is the probability of setting an input unit to zero, and $\odot$ denotes element-wise multiplication. Dropout helps improve the generalization of the model by ensuring that it does not become overly sensitive to specific inputs, thereby enhancing its robustness and performance on unseen data.

The padding layer Padding is a technique used to handle sequences of different lengths within a batch. It allows for the use of a single forward pass with a batch of sequences, all of which are initially of different lengths. The padding layer adjusts the length of shorter sequences by adding padding elements, ensuring that all sequences within the batch have the same length (Fig. 3.1H). Mathematically, the operation is expressed as:

$$\boldsymbol{z} = \upsilon_\alpha(\boldsymbol{x}, L_z); \quad \boldsymbol{x} \in \mathbb{R}^{L_x \times d_x}, \quad \boldsymbol{z} \in \mathbb{R}^{L_z \times d_x}; \quad L_x \leq L_z \tag{3.20}$$

$$\boldsymbol{x} = [x_1, x_2, \dots, x_{L_x}] \xrightarrow{\text{Padding through } \upsilon_\alpha} \boldsymbol{z} = [\underbrace{\alpha, \alpha, \dots, \alpha}_{L_z - L_x \text{ times}}, x_1, x_2, \dots, x_{L_x}] \tag{3.21}$$

where $\boldsymbol{x}$ represents the input sequence of length L_x and $\boldsymbol{z}$ represents the padded sequence of length L_z. The padding layer adds the necessary padding (denoted by α) to the sequence until it reaches the desired length L_z. Here, the sequence $\boldsymbol{x}$ is padded with α (the padding value) until the sequence length becomes L_z.

It is important to note that the padding layer is a "*trick*" layer. Its primary function is to allow the model to process batches of sequences with varying lengths. In other words, padding in itself is not a solution for managing variable sequence inputs on the scale of the whole problem — i.e. it cannot replace attention mechanisms coupled with a final pooling. Padding enables the attention and pooling mechanisms to be applied in parallel to a batch of sequences of different lengths, to benefit from the optimizations of numerical computations provided by the computer hardware.

To ensure that the padding does not interfere with the learning process, a padding mask is generated. This mask marks the padded elements in each sequence so that other layers, such as attention or pooling layers, can ignore the artificially added padding and focus only on the original, meaningful data in the sequence.

All these basic building blocks are assembled to create the final architecture of the model. This will be detailed in Section 5.5.1 of this work.

Interplay of attention, pooling, and padding The interplay of attention, pooling, and padding enables the DNN to effectively handle variable-length sequences. Attention mechanisms capture the contextual relationships within the sequence, pooling layers aggregate this information into a fixed-size representation, and padding techniques ensure that sequences of different lengths can be processed in parallel.

Together, these components allow the model to learn from variable-length sequences efficiently. The attention mechanism provides the flexibility to handle sequences of arbitrary length, the pooling layer ensures that the output is a fixed-size representation, and the padding technique enables parallel processing of batches of sequences.

3.2.3 Learning Techniques

This section introduces the different methods used in this work for training. First, we present the concept of gradient descent, which is the technique used to learn the model learnable parameters. Next, we introduce the random search method, which is used to optimize hyperparameters. Finally, we present LoRA and the concept of transfer learning, which extend the idea of a model and enhance the ability of the training process to generalize across tasks.

Learning by Gradient Descent

Training DNNs involves optimizing the model parameters θ to minimize a loss function $L[f_\theta]$. A widely used approach for this optimization is **gradient descent** (GD), an iterative algorithm that updates parameters by moving in the direction of the negative gradient of the loss function. This section outlines the fundamental principles of gradient descent and its variants.

The basic gradient descent algorithm Given a loss function $L[f_\theta]$, gradient descent updates the parameters θ iteratively:

$$\theta \leftarrow \theta - \eta \nabla_\theta L[f_\theta], \tag{3.22}$$

where η (the **learning rate**, a hyperparameter) controls the step size. The gradient $\nabla_\theta L[f_\theta]$ indicates the direction in which $L[f_\theta]$ increases the fastest, so taking steps in the opposite direction facilitates minimization. The use of *automatic differentiation* allows for the exact computation of gradients during execution time (Baydin et al., 2018).

Mini-batch gradient descent Computing gradients over the entire dataset — applying Eq. 3.3 — is computationally expensive and memory-intensive. Additionally, using the full dataset results in low-variance gradients, which can lead to overfitting, poor exploration of the parameter space, and ultimately degraded performance (LeCun et al., 2012).

A more efficient alternative is to use a subset of the dataset, B, known as a *batch*, for each update step. The batch size, $|B|$, is determined as a trade-off between efficient parallel computation and noisy gradient estimates, and it serves as a hyperparameter of the learning algorithm. At each iteration, a new batch is sampled from the dataset:

$$\theta \leftarrow \theta - \eta \frac{1}{|B|} \sum_{(x_i, y_i) \in B} \nabla_\theta \ell(f_\theta(x_i), y_i). \tag{3.23}$$

The problem of anisotropic and variable curvature Behind the classic formulation of gradient descent — Eq. 3.22 — lie two assumptions that are often incorrect in practical applications of DL: (1) the curvature of the loss landscape is constant, and (2) it is isotropic. The failure of these assumptions makes gradient descent optimization inefficient in practice.

Imagine a stone rolling down a hill; if the hill has a uniform slope and is perfectly smooth, the stone path is straightforward and predictable. However, if the hill has varying steepness and rough terrain (anisotropic and variable curvature), the stone path becomes erratic, inefficient and slow. To address this, imagine giving the stone some momentum, so as it rolls, it builds up speed, helping it overcome small bumps and maintain a straighter path downhill, this is connected to the *momentum* mechanism in optimizers. Furthermore, imagine equipping the stone with adaptive shock absorbers that adjust to the terrain, allowing it to handle rough patches more smoothly, much like how *adaptive learning rates* in optimizers like *Adam* and *AdamW* adjust the step size for each parameter, accounting for the varying curvature of the loss landscape. Finally, consider adding a small guiding force that keeps the stone from deviate too far off course, similar to the *regularization* mechanism in AdamW, which helps prevent overfitting by ensuring that the model parameters do not become too complex.

Variants of gradient descent, such as *Adam* (Kingma and Ba, 2017) and its improvement *AdamW* (Loshchilov and Hutter, 2019), incorporate these principles. We chose to use *AdamW* as the *optimizer* in this work. In practice, it comes with many hyperparameters, and interested readers can refer to Appendix F for a more detailed explanation of *AdamW*. In summary, this optimizer integrates momentum, adaptive learning rates, and regularization to improve convergence efficiency and reduce the risk of overfitting.

The vanishing gradient problem A frequent issue when training DNNs with GD is the phenomenon of *vanishing gradients* (Glorot and Bengio, 2010). This phenomenon occurs when the gradients calculated for updating the network parameters become extremely small, especially in the earlier (closer to the input) layers. When this happens, the weights of the first layers are almost never updated over iterations, preventing these layers from learning effectively. This problem is particularly pronounced in architectures using activation functions like the sigmoid or the hyperbolic tangent, whose derivatives are naturally small when the inputs take extreme values.

Many commonly used deep learning techniques are proposed as solutions to prevent or reduce the vanishing gradient problem. Among these methods are the use of unbounded activation functions like ReLU, residual connections, normalization layers, and specific parameter initialization schemes. The first three methods have been presented in this section; for initialization schemes, readers are referred to Glorot and Bengio, 2010 or Louppe, 2023[3].

How to Explore the Hyperparameter Space Using Random Search?

Since hyperparameters are not optimized by the optimizer, they are considered fixed during training. However, there are methods to explore different values for the hyperparameters to identify a good combination. As the model is often not differentiable with respect to these hyperparameters, techniques that diverge from gradient descent and are closer to trial and error are used. Among these methods, *random search* (Bergstra and Bengio, 2012) is a popular approach.

The idea behind random search is to choose a probability distribution for the hyperparameters and use it to sample different versions of the model. These versions are then trained on the training set and evaluated on the validation set. The best set of hyperparameters is the one that yields the best performance during evaluation.

Leveraging Existing Models to Improve Learning through Transfer Learning

Transfer learning is a branch of SL that focuses on extracting knowledge learned by a model trained on one task and incorporating it directly into another model so that it performs well on an identical or similar task.

In this work, transfer learning takes place when we wish to train the model on different CBMs. In practice, we hypothesize that there are links between the structure of the DICs of one CBM and another. The goal is to leverage what our model has learned from one CBM to improve its performance when trained on another. Transfer learning methods drastically reduce the data requirements when exploring a new task.

[3]Lecture *Training neural networks*

There are various solutions such as *fine-tuning* (Tam et al., 2022) or the use of *adapters*. The latter is the method we explore in this work, using *LoRA* adapters.

The idea behind an adapter is to exploit a pre-trained model (trained on task 1) by freezing it, meaning we no longer modify its learnable parameters, and using it as a backbone for the new task. On top of this backbone, we add new blocks that adaptively adjust the transformations. Unlike fine-tuning, adapters introduce additional parameters. However, the number of these parameters is often much smaller than the backbone itself, which allows for faster learning. Moreover, adapters can be used in a *modular* style, meaning that the model obtained with the adapters has not forgotten the first task, and simply disabling the adapters restores the original model. In a context where we have many different CBMs, this avoids having to store many large models and instead allows us to store a backbone and a set of small adapters that can be loaded as needed.

Low-Rank Adaptation Adapters LoRA adapters (Hu et al., 2021) are a type of adapter characterized by a very low number of additional parameters. To achieve this, LoRA relies on parameterized matrices with low rank: $\boldsymbol{A} \in \mathbb{R}^{d\times r}$, $\boldsymbol{B} \in \mathbb{R}^{r\times k}$ (Fig. 3.2). The adapter can be introduced in FC layers or self-attention layers by modifying each matrix multiplication by $\boldsymbol{W} \in \mathbb{R}^{d\times k}$ as follows:

$$\boldsymbol{W}x \xrightarrow{\texttt{LoRA Adapter}} \underbrace{\boldsymbol{W}x}_{\text{frozen}} + \underbrace{\frac{1}{r}x\boldsymbol{AB}}_{\text{learnable}}. \tag{3.24}$$

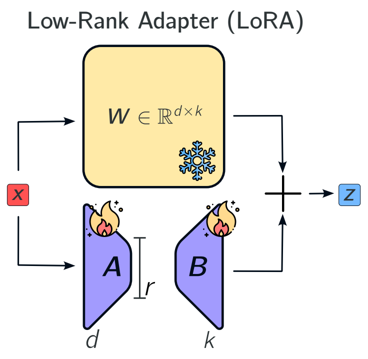


Figure 3.2: Illustration of the Low-Rank Adapter (LoRA) mechanism. This figure depicts the LoRA mechanism used to adapt a pre-trained model for new tasks efficiently. The LoRA adapter introduces a small number of additional parameters through low-rank matrices $\boldsymbol{A} \in \mathbb{R}^{d\times r}$ and $\boldsymbol{B} \in \mathbb{R}^{r\times k}$. These matrices are used to modify the matrix multiplication in fully connected or self-attention layers by adding a learnable component to the frozen pre-trained weights $\boldsymbol{W} \in \mathbb{R}^{d\times k}$. This approach allows the model to adapt to new tasks with minimal additional parameters, preserving the efficiency and performance of the original model while enabling modular and flexible adaptation.

Box VII – Building Blocks of Deep Learning

The Main Steps of a SL Pipeline Setup

The supervised learning process consists of several essential steps:

1. **Choice of features**: Determines the input representation for the model, often involving feature engineering and augmentation.
2. **Choice of architecture and hyperparameters**: Selects the model architecture and its hyperparameters, which influence the model capacity and flexibility.
3. **Choice of loss function**: Defines the measure of similarity between predictions and ground truth, guiding the optimization process.
4. **Choice of learning algorithm**: Specifies the method for updating model parameters, typically involving gradient descent.
5. **Tuning hyperparameters**: Uses techniques like random search to find optimal configurations for the model.
6. **Final training**: Optimizes the model parameters using the entire training dataset.
7. **Evaluation on test set**: Assesses the model ability to generalize to unseen data, providing a final performance metric.

Representation Learning Through Highly Parametrized Layers

Deep neural networks (DNNs) are designed to learn complex, successive mappings through multiple layers. Each layer applies a transformation parameterized by learnable weights, with these transformations stacked to form the complete network. The term "deep" refers to the use of multiple layers, allowing the network to approximate highly intricate functions. Training DNNs involves optimizing these parameters using techniques like gradient descent, where updates are guided by gradients computed via backpropagation. Mini-batch gradient descent is often preferred for balancing computational efficiency and noise reduction during optimization. Challenges such as vanishing gradients are addressed using methods like residual connections, normalization layers, and unbounded activation functions like ReLU.

An Encoder-Decoder Paradigm

The architecture used in this work follows an encoder-decoder paradigm. The encoder processes variable-length inputs, such as spike-time sequences, into a fixed-dimensional latent representation. This is achieved using architectures like Transformers, which excel at capturing long-range dependencies and handling sequence variability. The decoder then maps this latent representation into the desired output space, specifically the DIC values. This approach allows the model to handle diverse input lengths while generating fixed-size outputs. Unlike models based on summary statistics, this learned encoding enables the model to extract task-relevant features without imposing predefined constraints:

$$\underbrace{\boldsymbol{x} \in \mathbb{R}^{*\times 1} \xrightarrow{\text{Encoder } \mathcal{E}} z_{\text{latent}}}_{\text{Projection into latent space}} \underbrace{\in \mathbb{R}^{d_{\text{latent}}} \xrightarrow{\text{Decoder } \mathcal{R}_{\text{DICs}}} \boldsymbol{y} = \begin{bmatrix} g_{\text{s}} \\ g_{\text{u}} \end{bmatrix} \in \mathbb{R}^2}_{\text{Mapping to DICs space}}.$$

Transfer Learning and the LoRA adapters

Transfer learning is employed in this work to leverage knowledge from pre-trained models, reducing data and computational costs when adapting to new tasks. Specifically, Low-Rank Adaptation (LoRA) adapters are used to modify pre-trained models efficiently. LoRA adapters introduce low-rank matrices to adaptively adjust the model transformations, adding minimal additional parameters. This modular functionality allows for rapid adaptation to new tasks without retraining the entire network. By storing a backbone model alongside small, task-specific adapters, this approach ensures efficient storage and deployment while preserving the original model capabilities.

Chapter 4

Literature Review — Automated Generation of Conductance-Based Models with Targeted Activity

This chapter focuses on reviewing the literature concerning procedures aimed at automating the generation of conductance-based models, with the possibility of conditioning on target electrical activity. The objective is to clearly position this Master Thesis to clarify its novelty while pointing towards alternative solutions. In addition to providing several pointers, an effort is made to identify the strengths and weaknesses of each.

Among the comparison criteria, the following points are considered:

- **Interpretability:** The goal behind conductance-based models is to remain within a relatively interpretable framework. Therefore, we will consider whether the proposed methods offer an interpretable framework.
- **Support for degeneracy:** As previously explained, degeneracy is a central concept in the analysis of neural activities. We will consider whether the methods support degeneracy by providing a set of parameter sets associated with an activity or provide only a point estimate of the conductance values.
- **Descriptive input of activity:** In this work, activity is characterized solely through spike times, which is widely accessible information even in the study of large neural networks. We will consider whether the presented methods require more or less information as input.
- **Computational cost:** If possible, we will highlight the computational cost required for inference and, if applicable, for training the model. This also includes the need for training data.
- **Performance:** Finally, where possible, we will emphasize whether the obtained performances are satisfactory or not; and whether the method is mathematically-grounded or not.

Search in a Database

The observation of the difficulty in hand-tuning conductance values is not recent and was already highlighted by Prinz et al., 2003. In this paper, the authors propose an alternative based on searching a database of pre-simulated instances classified by activity type and statistical summary. The database contains 1.7 million instances of the STG model and can be explored to find appropriate conductance values that yield activity close to the target activity.

Although this database represents an initial effort, note that it is inherently fixed, and the already existing instances are the only outputs this method can produce. As highlighted in another paper by Golowasch et al., 2002, interpolation in the conductance space is not a reliable technique. Thus, the database is limited to its training data. The inference cost is close to zero, and the training cost essentially reduces to the cost of simulating the instances. Degeneracy is only partially accounted for, as it is possible to obtain multiple instances that match the target activity, always within the dataset limits. Beyond the obvious limitations of a finite database, this paper illustrates the problem of methods based on random (naive) search. Specifically, one is limited to blind sampling of the conductance space and post-hoc filtering to observe degeneracy. Finally, the database allows for observing relationships between conductance values and descriptive metrics of activity. However, in practice, these relationships provide limited interpretability of the model, which strongly resembles the expertise required in hand-tuning.

Ultimately, the database is limited to the STG model, and extending it to other types of CBMs requires completely restarting the work.

Genetic Algorithm

An alternative to the fixed database is proposed by Druckmann et al., 2007, which involves exploring the conductance space using a genetic algorithm to solve a multi-objective optimization problem. The proposed method involves generating a random population, simulating it to evaluate whether the instances exhibit activity similar to the target one. Similarity is evaluated based on multiple criteria — making it a multi-objective optimization problem — and the instances closest to the target activity are selected for the next iteration. In each iteration, the discarded instances are replaced by mutated versions of the selected instances, i.e., instances with re-sampled values for some of the conductances. This process of simulation, evaluation, selection, and mutation is repeated until a population with activity close to the target activity is obtained.

This method has advantages over classical databases since it is not restricted to a fixed set of data. However, it is computationally expensive because the procedure must be repeated for each target activity. In other words, there is no separate training and inference phase; the model must be retrained for each target activity. A solution to optimize the cost of these numerous simulations in a multi-machine (multiprocessing) optimization context and the intensive use of vector operations is proposed by Rossant et al., 2011. In addition, convergence is not guaranteed and can be slow. Moreover, the limited interpretability that the database allowed is completely lost. Notably, the authors propose analyzing conflicting objectives as interpretable elements of the model but we find this interpretation quite weak.

The model inputs — i.e., the objectives — are not solely based on spike times. The authors use additional information extracted from the trace, such as the overshoot value of the action potential or its width, which may not be available experimentally.

Finally, adding a term that favors diversity among instances within the same population allows for directly accounting for degeneracy. This results in different instances with similar activity but different conductance values. The method is illustrated on two cortical interneuron models and can be directly applied to other model types.

Analytical and Numerical Method to Invert Hodgkin-Huxley Models

In his thesis, Shepardson, 2009 proposes a numerical procedure aimed at '*inverting*' CBMs — i.e., extracting the corresponding parameters from the solution of the ODEs system. In addition to the procedure, analytical developments prove the convergence of the method to a perfect reproduction of the target activity in the case of the simplest CBMs — for example, the classic Hodgkin-Huxley model or the DA model of this work. However, for the STG model, convergence is not guaranteed due to explicit calcium dependence modeling.

The proposed method involves integrating the ODEs locally using a stable numerical scheme while keeping the conductances unknown, resulting in a linear system that can be solved in the least squares sense at multiple time values simultaneously. In practice, this method has the advantage of being supported by theoretical elements. However, note that the procedure requires the complete trace of the activity, and the influence of noise is not studied in the original work, although it would significantly impact the method results and is inherent to true experimental conditions. The model cost is relatively low since no training is necessary, and inference is reduced to solving a few small linear systems. The method is not interpretable, and degeneracy is not handled since the obtained estimation is a single instance and not a population. Although using initial conditions for the algorithm allows convergence to different estimations, it is a tedious process.

Huys et al., 2006 proposed a similar method that scales very well with the number of parameters (up to 10^4 parameters in their study). The method relies on spatiotemporal measurements of membrane potential and is demonstrated to be effective on synthetic data argued to be similar to *in vitro* data. It is very cost-effective in terms of data for inference. There is no training, technically speaking. However, the method cannot be exploited for discontinuous measurements of potential — such as spike time sequences — and this strongly limits its effectiveness. Finally, although noise is encoded in the model, no analysis is provided on its impact on the procedure efficiency. In practice, Huys et al., 2006 relies on solving a non-negative linear convex problem. Convexity implies that only one solution exists for the optimization, and thus degeneracy is not handled by this method. Interpretability arises through the linear filters that weight the unknown parameters to be estimated, which can guide experts in understanding the model decisions. However, interpretability remains fairly limited.

Burghi et al., 2021 introduces a method for estimating biophysical parameters from voltage-clamp-like data using tools from nonlinear system identification and control theory. Their approach is based on the Prediction Error Method, adapted for discrete-time CBMs under closed-loop feedback, similar to conditions in voltage-clamp experiments. This setup ensures that the system inverse dynamics are exponentially contracting, a property supported by contraction theory, which guarantees global stability and consistent estimation even with biologically realistic noise, provided the reference signals are persistently exciting.

This method is computationally efficient, as it involves solving small linear least squares problems without requiring a training phase. The fact that this method uses data from voltage-clamp-like experiments makes it more interpretable and closer to classical neurosciences laboratory techniques. While the method does not explicitly address degeneracy, varying the reference signals can help explore different solutions. It has been shown to work well with models like Hodgkin-Huxley and Connor-Stevens, where estimation error decreases with longer and more informative input traces. However, it is limited to scenarios where full membrane voltage traces are available under stabilizing control and does not support spike-train-only or passive recordings.

In a related work, Burghi et al., 2022 present an adaptive observer framework for real-time estimation of state variables and parameters in CBMs. This observer uses recursive least squares combined with contraction-based dynamics to ensure convergence despite uncertainty or time-varying parameters. The adaptive structure allows for continuous updating of parameter estimates, making it suitable for closed-loop experiments or embedded systems. It can handle some level of degeneracy due to its sensitivity to initial conditions and ongoing input variation, though it does not explicitly model or quantify degenerate parameter sets.

The adaptive observer requires continuous voltage and input current traces, and supports online inference with minimal computational overhead. It has been validated on single-compartment Hodgkin-Huxley models and network-level half-center oscillators, demonstrating accurate recovery of membrane potential, gating variables, and conductance parameters, even under noisy or mismatched initializations.

Both Burghi et al., 2021, 2022 are notable for their strong mathematical foundation and the provision of convergence proofs through contraction theory. However, they require very specific data types, namely continuous traces from closed-loop experiments, that are not always available or practical to collect.

Fitting of Reduced Models

In contrast, Rossant et al., 2011 propose using reduced or hybrid models to reproduce traces or spike trains recorded in laboratories. The paper explores the use of the classic Hodgkin-Huxley model or an integrate-and-fire model (for which we can build easily maximum likelihood estimation e.g. following Paninski et al., 2004). They apply this idea and demonstrate that it is possible to obtain a good match with the target activity. Without relying on a specific parameter fitting technique, the idea of using simpler models itself presents an explorable avenue that can reduce costs and, in some cases, the amount of input information required to achieve an acceptable match. The trade-off, however, is the loss of direct biological interpretability if the model used is less aligned with biological mechanisms. Under this same observation, it is worth questioning whether degeneracy remains a central element. Such methods seem better adapted for large network approximations.

However, in practice, this approach does not provide a real solution for generating CBMs, as it relies on simplified models that do not fully capture the complexity of biological systems.

Particle Filters (Sequential Monte Carlo Methods)

Many studies propose obtaining non-parametric approximations of the distributions of neural model parameter values, given a target activity, using particle filters (Ditlevsen and Samson, 2014; Huys and Paninski, 2009; Meng et al., 2011, 2014). These works are based on the same idea of using a generative model to simulate samples (or particles) from the parameter space. These particles are weighted according to their ability to reproduce the observed target activity. Ultimately, each of these particles, along with its weight, can be used as an approximation of the posterior distribution of the parameters. Particle filters rely on the idea that observations are sequentially related to latent (internal) components that cannot be directly observed, all within a probabilistic framework that inherently models noise.

In Meng et al., 2011, 2014, the observations are spike times; in Meng et al., 2011, the method is applied to hybrid models and a reduced version of the Hodgkin-Huxley model, with estimated parameters including both conductances and current values. In Meng et al., 2014, the method is extended and applied to data recorded *in vitro* in laboratories. In Ditlevsen and Samson, 2014, the inputs are point values of the trace rather than spike times, and the model used is a stochastic model, with *in vitro* data demonstrating practical applicability. Finally, Huys and Paninski, 2009 proposes using numerous intra- and extracellular measurements as observations. The method is applied to various CBMs with varying numbers of conductances, and while the results are very consistent for low-dimensional models (4 conductances), performance is notably biased for high-dimensional models (8 conductances), i.e. as such, the method lacks capacity for good generalization.

These papers highlight that particle filters can be effective methods for diverse inputs and variable models. However, they also show that this method scales poorly with dimensionality — a problem for CBMs, which tend to be of increasing dimensionality; and in opposition with the needs of target users who want precise (complex) models — and requires a high computational cost. The cost of inference and training are combined since, similar to genetic algorithms, the model must be retrained for each new activity of interest. Nevertheless, particle filters allow for an approximation of the joint posterior distribution of conductances, which is a considerable advantage for considering degeneracy. Finally, interpreting the model beyond observing the results seems complicated.

Simulation-Based Inference using DNNs

Deep Learning techniques and *Simulation-Based Inference* methods (Cranmer et al., 2020) have been shown to be effective in high-dimensional scenarios. For inferring parameters of CBMs, Gonçalves et al., 2020 demonstrated the power of these methods. In their paper, they show how training a DNN (in this study, a masked autoregressive flow) allows for determining complete distributions in the parameter space for isolated neurons, small synthetic circuits, and even *in vitro* setups — using only spike times. The inference of a complete joint distribution is a major advantage of this method and allows for directly handling degeneracy. Additionally, the parametric nature of these distributions enables analyses that go beyond examining conductance values. For example, using their method, the authors were able to rediscover compensation mechanisms.

Despite the advantages and impressive results of Gonçalves et al., 2020, note that the method is highly data-intensive. For the STG model circuit, 18.5 million simulations were required to implement the full pipeline. This intensive data requirement poses a challenge for extending the model to numerous CBMs. However, it is worth noting that, as in this work, transfer learning methods could potentially mitigate this issue. On the other hand, the computational cost is primarily associated with training, and inference is relatively inexpensive.

This high data demand is not unique to their approach but rather a general characteristic of DNN-based simulation inference methods. For instance, Saghafi et al., 2024 used conditional generative adversarial network (cGANs) to infer distributions over only five maximal conductances, yet required over 3 million simulations to train the model.

Finally, these methods lack interpretability due to the black-box nature of DNNs.

Perspective with the Method of This Work

This work positions itself midway between highly interpretable methods and fully black-box approaches. Similar to Gonçalves et al., 2020, our approach uses *simulation-based inference* to train a DNN, which involves creating synthetic datasets from mechanistic conductance-based models to infer parameters from recorded neuronal activity. However, our method stands out by incorporating the framework of Dynamic Input Conductances (DICs) which is mathematically-grounded.

The DICs provide a more interpretable representation, even in high-dimensional conductance spaces, with results interpretable from a global feedback perspective. This allows the model to stay grounded in biophysical reality while remaining practical for inference. Additionally, this approach requires less data than deep learning methods like masked autoregressive flows, as it doesn't need to learn a full posterior distribution directly. Instead, inference is done in the DICs space, and conductance distributions are derived using established theoretical and numerical results (notably from Fyon et al.).

Unlike highly interpretable methods that often provide point estimates without accounting for degeneracy, our framework can recover a complete population of instances that produce similar activity patterns. At the same time, it avoids the heavy computational demands of methods like genetic algorithms and particle filters, which require redoing training from scratch for each new activity pattern.

Finally, our method can be used in a wide range of experiments since it relies only on the spike time measurements and does not require the full voltage trace. Moreover, we prove it to be robust to noise and very fast in inference, making our solution well adapted for daily laboratory usage.

Chapter 5

Development and Methodology

This chapter presents the core methodological framework and development processes underpinning this Master Thesis. Building upon the theoretical background introduced in the preceding chapters, this chapter provides a detailed exploration of the practical steps taken to achieve the objectives of this work. It covers the preliminary considerations, the development of key algorithms, and the methodological approaches employed to generate datasets, design model architectures, and implement training strategies. Additionally, it discusses the transfer learning techniques used to adapt the model to different conductance-based models.

Structure of the Chapter

The first section, **5.1 Preliminaries**, sets the stage by outlining the essential background and considerations that guided the methodological choices.

The second section, **5.2 Iterative Compensation Algorithm**, introduces a novel algorithm designed to address the limitations of the existing DICs compensation method, particularly in handling complex models like the STG.

The third section, **5.3 Reachability and the Limits of Biophysically Plausible Compensation**, explores the theoretical and practical constraints of compensation in the context of neuronal degeneracy.

The fourth section, **5.4 Dataset Generation**, provides a comprehensive overview of the procedures used to create the datasets necessary for training and validating the deep learning models. This includes the generation of degenerate populations and the exploration of the datasets to ensure their quality and relevance.

The fifth section, **5.5 Model Architecture and Training Strategy**, delves into the design and implementation of the deep neural networks used in this work, including the architectural choices, training strategies, and hyperparameter explorations.

Finally, the sixth section, **5.6 Transfer to the Dopaminergic Neuron Model**, discusses the application of transfer learning techniques to adapt the model to different CBMs, ensuring its versatility and robustness.

By the end of this chapter, readers should have a clear understanding of the methodological framework and development processes behind the results and contributions of this Master Thesis.

Note on Focus and Scope

To keep the core of the thesis concise and easy to read, the discussion will center around the STG model, except where explicitly stated otherwise. References to appendices will be made to discuss the DA neuron model. The methodologies and algorithms presented here are quite general and can be easily transposed to new CBMs.

5.1 Preliminaries

This section aims to illustrate the general methodology and the preliminary analyses necessary to apply the method of this thesis to any CBM. These are summarized in the remainder of this section, with pointers to the corresponding appendices provided.

The main objective of this work is to establish a deep learning pipeline that generates populations of degenerate CBMs through the theory of DICs, using only spike time records. The general methodology can be divided into several distinct tasks:

(1) Regardless of how the target DIC values are determined, we must be able to generate populations from these values — i.e., to construct a vector of maximum conductance values for each instance. The generation procedure must be both precise and reliable. The purpose of Section 5.2 is to present the new method developed in this work that makes the procedure precise. The purpose of Section 5.3 is to present the new perspectives identified to make the generation reliable. By combining these two sections, we are able to choose the set of conductances to be compensated and to generate populations that exhibit the desired DIC values. This combination is necessary both for generating the dataset and for actually generating the conductance values of the degenerate populations from the predictions of our pipeline during inference.

The generation procedure requires providing initial conductance distributions and a threshold potential. The purpose of Appendix E is to justify why we introduce two different distributions, one for the preliminary analyses and one for the generation procedure, as well as a numerical methodology and justification elements for identifying the value $V_{\text{th}} \approx -51\,\text{mV}$ used in this work for the STG model.

(2) Generate the dataset necessary for training the pipeline. This step involves simulating a large number of instances associated with different target DIC values. Section 5.4 presents the details of the procedure.

In this work, we limit ourselves to a portion of the DICs space. Appendix E justifies the choice of this portion, which widely covers the DIC values naturally associated with the chosen distributions. For each target DIC values sampled in this range, we will need to generate a population of several instances and simulate them.

The simulation of the different instances requires numerically integrating the associated system of ODEs. Appendix B.3 details the solver used and the use of multi-CPU to accelerate the generation. Since the simulation of CBMs provides the full trace, Appendix D explores the method used to transform these into sequences of spike times to illustrate the targeted laboratory conditions. We also detail the definitions of various descriptive metrics of neuronal activity, such as spiking frequency or the number of spikes per burst; these are used during model training as will be detailed in Section 5.5. In this appendix, we also detail the classification of neuronal activities and introduce the van Rossum distance, which is a quantitative metric used to compare sequences of spike times in both spiking and bursting in a unified manner.

(3) Establish the architecture of the deep learning pipeline. That is, the arrangement of different building blocks that process the sequence of spike times received as input into slow and ultra-slow DIC values. Section 5.5 presents the details of this architecture.

5.2 The Iterative Compensation Algorithm

5.2.1 Context

In Section 2.4 of this work, we presented the computationally efficient method proposed by Fyon et al., 2024 for generating degenerate populations from DIC values following Algorithm 1. However, we identified a limitation of this method for models such as the STG model, which exhibit explicit ionic dependencies: the compensation algorithm cannot be exactly expressed as a linear system. In this context, the method is nonlinear since, in the compensation relation (2.15), the sensitivity $\boldsymbol{S}$ directly depends on the value of Ca_∞, which itself depends on $\bar{g}_{\mathrm{CaS}}$ and $\bar{g}_{\mathrm{CaT}}$. If these conductances are among the compensated ones, we have $\boldsymbol{S} = \boldsymbol{S}(\bar{\boldsymbol{g}}_{\mathrm{comp.}})$ that cannot be evaluated *a priori*. This corresponds to the classical formulation of a system as follows:

$$\boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} = \boldsymbol{b}; \quad \text{which is nonlinear when considering } \boldsymbol{x} \text{ as the unknown.} \tag{5.1}$$

The approximate method proposed by Fyon et al., 2024 to transform the system (5.1) into a linear approximation relies on a default value $\boldsymbol{x}^*$, which is used to evaluate $\boldsymbol{A}(\boldsymbol{x})$. In practice, they rely on a default value of $\bar{g}^*_{\mathrm{CaS}}$ used to determine an approximate value of Ca_∞. In other words:

$$\texttt{Method of Fyon et al., 2024:} \quad \boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} = \boldsymbol{b} \xrightarrow{\boldsymbol{A}(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}^*}=\boldsymbol{A}^*, \quad \tilde{\boldsymbol{x}}\approx\boldsymbol{x}} \boldsymbol{A}^*\tilde{\boldsymbol{x}} = \boldsymbol{b} \tag{5.2}$$

Since the obtained result $\tilde{\boldsymbol{x}}$ is only an approximation of the exact solution $\boldsymbol{x}$ of the nonlinear system (5.1), the residual is nonzero. That is, $\boldsymbol{A}(\tilde{\boldsymbol{x}})\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}} \neq \boldsymbol{b}$, leading to $\|\boldsymbol{r}\| = \left\|\tilde{\boldsymbol{b}} - \boldsymbol{b}\right\| > 0$, where $\|\boldsymbol{r}\|$ denotes the residual. From the perspective of the generation method, this means that the target DICs are not exactly achieved, as illustrated in Figure 2.15.

The method (5.2) has the advantage of being simple. It reduces the problem to solving two small linear systems — recall that in step 1 of Algorithm 1, $\boldsymbol{A} \in \mathbb{R}^{3\times3}$, and in step 2, $\boldsymbol{A}^* \in \mathbb{R}^{2\times2}$. However, we hypothesize that the residual error obtained by following procedure (5.2) may lead to excessive variability in the DICs of different instances within the same population. Such variability could be undesirable in control problems or could impact the degenerate proximity of the generated populations, which is particularly problematic in this work.

5.2.2 Proposed Alternative Procedure

As a major contribution of this work, we propose an alternative method to reduce the residual error — i.e., to more accurately impose the DIC values within the generated populations. This improvement in precision comes at the cost of a slight additional computational cost. However, we argue that this cost remains negligible compared to traditional degenerate population generation methods and can be adjusted based on precision requirements.

The proposed alternative approximation method for system (5.1) is called the **Iterative Compensation Algorithm**. It consists of two steps: the first is optional but provides an almost cost-free accuracy improvement, while the second is the core of the iterative compensation algorithm.

Step 1: A better a priori default value Observing that $\tilde{\boldsymbol{b}} \neq \boldsymbol{b}$ arises because $\tilde{\boldsymbol{x}} \neq \boldsymbol{x}^*$, a better *a priori* estimate of $\boldsymbol{x}^*$ — which is a fixed value in the method of Fyon et al., 2024 — can yield a smaller residual. If a dataset $\mathcal{T} = \{(\boldsymbol{x}_i, \boldsymbol{b}_i)\}_{i=1}^{T}$ exists or can be easily generated, we can construct an approximation model $\boldsymbol{x}^* = \hat{\boldsymbol{f}}_{\mathcal{T}}(\boldsymbol{b})$. If the model computational cost is low, the following method (5.3) provides a direct improvement over (5.2):

$$\texttt{Target-dependent Prior Method:} \quad \boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} = \boldsymbol{b} \xrightarrow{\boldsymbol{A}(\boldsymbol{x})|_{\boldsymbol{x}=\hat{\boldsymbol{f}}_{\mathcal{T}}(\boldsymbol{b})}=\boldsymbol{A}^*, \quad \tilde{\boldsymbol{x}}\approx\boldsymbol{x}} \boldsymbol{A}^*\tilde{\boldsymbol{x}} = \boldsymbol{b} \tag{5.3}$$

For the STG model, we can construct a dataset $\mathcal{T}_{\text{STG}} = \left\{ \left(\text{Ca}_{\infty,i}, \boldsymbol{g}_{\text{DICs},i} \right) \right\}_{i=1}^{T}$. To maintain low computational costs, we can use a linear approximator, such as:

$$\text{Ca}_\infty \approx \hat{f}_{\mathcal{T}_{\text{STG}}}(\boldsymbol{g}_{\text{DICs}}) = b + w_1 g_{\text{s}} + w_2 g_{\text{u}}$$

The dataset $\mathcal{T}_{\text{STG}}$ is built by sampling $M = 10000$ instances of the STG model, computing the associated $\boldsymbol{g}_{\text{DICs}}$, and applying relation (2.18) for each instance. A linear regression minimizing the squared errors is then performed. Figure 5.1 illustrates the procedure. The numerical results indicate greater sensitivity to g_{s} than to g_{u}:

$$\text{Ca}_\infty \approx \hat{f}_{\mathcal{T}_{\text{STG}}}(\boldsymbol{g}_{\text{DICs}}) = -0.0299 g_{\text{s}} - 0.0056 g_{\text{u}} + 0.5679 \tag{5.4}$$

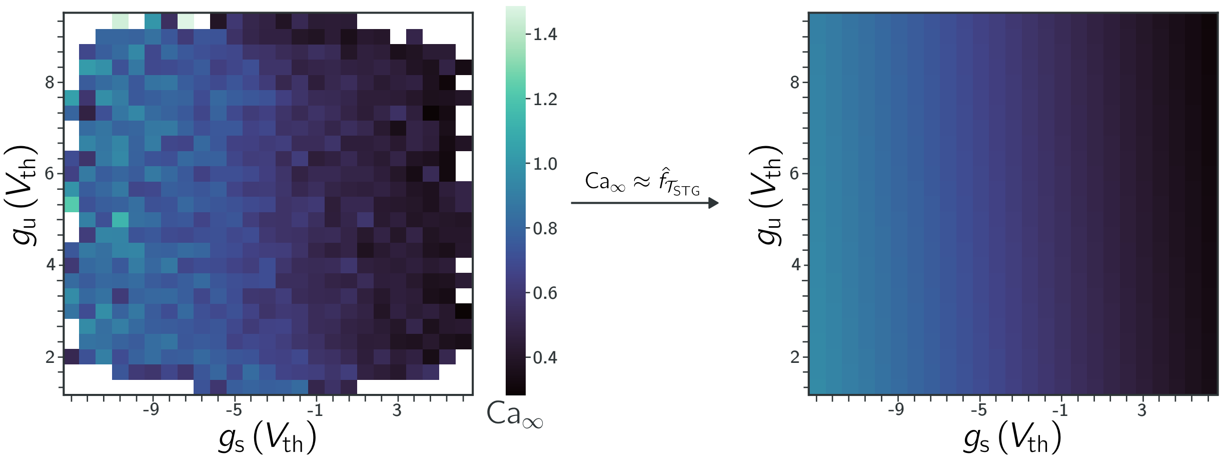


Figure 5.1: Approximate linear mapping of equilibrium calcium concentration in the STG model. The left panel displays a binned statistic of the observed equilibrium calcium concentration (Ca_∞) as a function of the DICs ($g_{\text{s}}, g_{\text{u}}$), computed from $M = 10,000$ sampled instances. The right panel presents the corresponding linear approximation $\text{Ca}_\infty \approx \hat{f}_{\mathcal{T}_{\text{STG}}}(g_{\text{s}}, g_{\text{u}})$, obtained via least-squares regression. A linear model is chosen due to its low computational cost, making it well-suited for efficient population generation. The transformation illustrates how the linear model smooths the observed data while capturing the dominant trend.

Step 2: A fixed-point iteration approach The problem (5.1) can be interpreted as finding a fixed point of the system $\boldsymbol{x} = \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{A}^{-1}(\boldsymbol{x})\boldsymbol{b}$. Under specific conditions where $\boldsymbol{g}$ is a contraction mapping, the iterative procedure $\boldsymbol{x}_{k+1} = \boldsymbol{g}(\boldsymbol{x}_k)$ converges to the unique fixed point — i.e., the solution of (5.1) — for any initial value $\boldsymbol{x}_0$ (Banach, 1922, Burden and Faires, 2010, Chapter 10).

In this work, we do not formally prove that we are dealing with a contraction mapping, but we provide empirical evidence that the method works and helps reduce residuals. Finally, this second step can be applied multiple times. Starting from an initial approximation $\boldsymbol{x}_0$, we perform:

$$\texttt{Iterative Compensation Method:}\quad \boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} = \boldsymbol{b} \xrightarrow{\boldsymbol{A}(\boldsymbol{x})|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}_k} = \boldsymbol{A}_k^*, \quad \tilde{\boldsymbol{x}}_{k+1} \approx \boldsymbol{x}} \boldsymbol{A}_k^* \tilde{\boldsymbol{x}}_{k+1} = \boldsymbol{b} \tag{5.5}$$

The complete procedure used in this work to generate datasets, which we have called **the iterative compensation algorithm**, is summarized in Algorithm 4 (Appendix C). The estimator model used for Ca_∞ is the previously introduced linear approximator. In practice, the iterative steps alone would probably have been sufficient. The innovative approach compared with Algorithm 1 lies in the *Iterative Compensation Step* (Algorithm 3, Appendix C) that we use if either $\bar{g}_{\text{CaS}}$ or $\bar{g}_{\text{CaT}}$ must be compensated (then *Nonlinear compensation is detected*).

The additional computational cost introduced by step two is adjustable. The more iterations you perform, the more you increase it, to the benefit of the accuracy of the generation. Note that the number of linear systems to be solved is linearly growing with the number of iterations. In practice, as the systems are very small, it's perfectly acceptable to perform several iterations, even for very large populations. If we assume that the computational cost of the approximate model for the initial guess is negligible — or that no such model is used — and that evaluating $\boldsymbol{A}(\boldsymbol{x})$ is also computationally inexpensive, then the time complexity of Algorithm 3 is $\mathcal{O}(Kn_{\boldsymbol{A}}^3)$, with K the number of iterations and $n_{\boldsymbol{A}}$ the size of $\boldsymbol{A} \in \mathbb{R}^{n_{\boldsymbol{A}} \times n_{\boldsymbol{A}}}$.

5.2.3 Comparison of Methods

We provide evidence that the iterative compensation method yields better results and more accurately enforces the target DIC values during population generation. Three methods are applied and compared:

- The first method follows the approach introduced by Fyon et al., 2024, using the default value $\bar{g}_{\mathrm{CaS}} \approx 10\,\mathrm{mS\,cm^{-2}}$.
- The second method, referred to as *target-dependent*, relies on the linear approximation of the calcium equilibrium value based on the target DIC values (**Step 1** and Eq. (5.4)).
- The third method applies the *iterative compensation algorithm* for different numbers of iterations (**Step 1** <u>and</u> **Step 2**). The values $K \in \{1, 2, 3, 5, 10\}$ are explored.

The evaluation process is based on computing the average residual $\|\boldsymbol{r}\|$ using the L^2-norm. The average is taken over $M = 250$ instances for each population, compensating $\bar{g}_{\mathrm{A}}$ and $\bar{g}_{\mathrm{CaS}}$. Target DICs are sampled from a uniform distribution over the grid:

$$(g_{\mathrm{s}}, g_{\mathrm{u}})_p \sim \mathcal{U}([-20, 20] \times [0, 20]), \quad p \in \{1, 2, \ldots, P\}.$$

DIC values that cannot be reached by compensating $\bar{g}_{\mathrm{A}}$ and $\bar{g}_{\mathrm{CaS}}$ are excluded. Specifically, if any instance in a population results in conductance values that are not strictly positive, that population is removed from the evaluation. After filtering, a total of $P^* = 1633$ populations remain from the initial $P = 5000$ populations.

Figure 5.2 presents boxplots of the residual distributions for these P^* populations, comparing all evaluated methods. Table 5.1 reports the mean residual values across the P^* populations for each method.

This numerical evaluation highlights that the proposed iterative method significantly improves the accuracy of generated populations. The performance gain comes primarily from the iterative refinement process rather than the target-dependent default value. The method of Fyon et al., 2024 results in residuals reaching up to 3.5, with an average around 1 — a substantial error given the grid scales. Using the iterative algorithm with $K = 5$ iterations, we reduce the mean distance in DIC space to below 0.1, with an empirical maximum of 0.75. Higher accuracy can be achieved by increasing K, but $K = 5$ was chosen as a good trade-off between precision and computational cost for dataset generation.

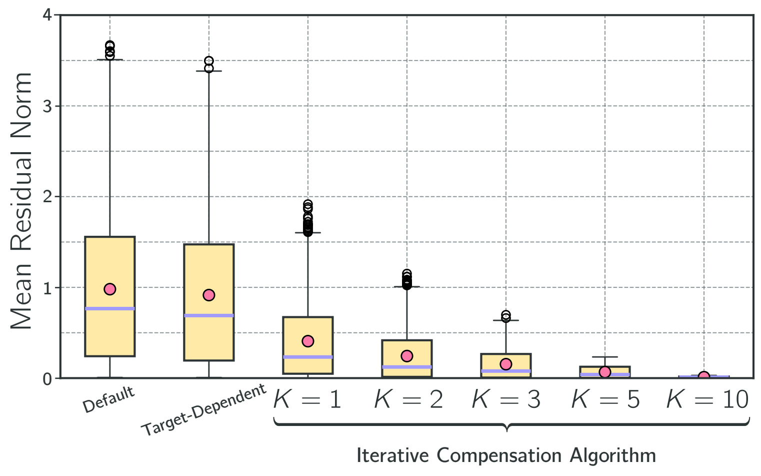


Figure 5.2: Residual distributions for different population generation methods. Boxplots showing the distribution of residuals (measured using the L^2- norm) across $P^* = 1633$ populations of $M = 250$ instances, for various generation methods. The median is indicated by the horizontal line within each box, while the mean is represented by a marker. Outliers are displayed as individual points. The iterative compensation method significantly reduces residuals compared to the default (Fyon et al., 2024) and target-dependent approaches, with increased iterations further improving accuracy.

Method	**Mean Residual** (L^2- norm)
Default (Fyon et al., 2024)	0.9845
Target-dependent (*linear equilibrium calcium fit*)	0.9151
Iterative Compensation Algorithm, $K = 1$	0.4072
Iterative Compensation Algorithm, $K = 2$	0.2434
Iterative Compensation Algorithm, $K = 3$	0.1536
Iterative Compensation Algorithm, $K = 5$	0.0667
Iterative Compensation Algorithm, $K = 10$	0.0106

Table 5.1: Mean residuals for different efficient population generation methods. This table compares the mean residuals (measured using the L^2- norm) for various compensation methods. Lower values indicate better accuracy in enforcing the target DICs. The iterative compensation method, particularly with increased iterations, significantly reduces the residual error compared to the default (following Fyon et al., 2024) and target-dependent only approaches. The population size is $M = 250$, and the number of compared populations is $P = 1633$.

5.3 Reachability and the Limits of Biophysically Plausible Compensation

5.3.1 Context

A second observation we made when presenting the population generation method from Fyon et al., 2024 was that certain choices of target DICs and their corresponding compensated conductance sets sometimes led to biophysically impossible results, specifically negative maximal conductance values. This phenomenon was illustrated in Figure 2.16A.

To our knowledge, this phenomenon has not been analyzed through the lens of DICs theory. However, its implications are particularly relevant to our study, since they indicate the extent to which we are able to generate populations within the DICs space by compensating for a given set. The analysis we present in this section is a major contribution of this work, offering novel insights into this issue. We propose a detailed examination and present various results and reflections on the topic.

Terminology

We introduce the concept of **reachability** of a point in the DICs space through the compensation of a given set of conductances. A point is said to be reachable if the generation procedure described in Algorithm 4 effectively produces — with high probability — populations with positive maximal conductance values.

The phrase *with high probability* reflects the stochastic nature of the generation procedure, which arises from the initial sampling following the distribution $\mathcal{D}_{\text{generation}}$ (Appendix E). According to the generation method, the factors that directly influence reachability are the distribution of conductances, $\mathcal{D}_{\text{generation}}$; the sensitivity matrix, $\boldsymbol{S}(V_{\text{th}})$; and the values of the DICs used to observe spontaneous behavior, $\boldsymbol{g}_{\text{DICs,spont.}}$.

The reachability analysis we propose focuses on the influence of the compensated conductance set while keeping $\boldsymbol{g}_{\text{DICs,spont.}}$ and $\mathcal{D}_{\text{generation}}$ fixed. This choice reflects the relevance of reachability in the context of this study: we aim to generate degenerate populations for a wide range of points in the DICs space. The primary objective is to identify a heuristic for selecting which conductances to compensate when attempting to reach a given point in the DICs space using our generation procedure.

In other words, reachability analysis in this work is conducted as a necessary step to overcome a technical challenge in population generation. However, this section will also discuss broader implications of reachability analysis and explore perspectives that extend beyond the technical heuristic.

Why Can Compensation Fail?

We identify two factors that can lead to compensation failure: *numerical instabilities* and *opposite sign sensitivity*.

Numerical instabilities The first factor is difficult to predict in advance. In practice, compensation does not fail under the criterion of negative conductances, but rather due to numerical overflow or underflow. Two signs indicate this issue: conductance values remain positive but become extremely large (e.g. of the order of 10^{22}), and more importantly, *a posteriori* computation of the DICs reveals that the target values were not actually reached.

The source of this instability lies in the inversion of the matrix $\boldsymbol{A}$ when solving the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, particularly in cases where $\boldsymbol{A}$ is nearly singular. This situation arises when the compensated conductances do not sufficiently influence all relevant timescales (Fyon et al., 2024). Ideally, compensation should be impossible in such cases, but numerical errors lead to inconsistent results.

Opposite sign sensitivity The second factor is not numerical but rather a consequence of the DICs theory. Compensation can fail even when the set of compensated conductances effectively influences the relevant timescales — this is illustrated in Figure 2.16A. The origin of this failure can be understood by examining the following formulation of compensation:

$$\boldsymbol{S}_{\text{comp.}}\Delta\bar{\boldsymbol{g}}_{\text{comp.}} \approx \Delta\boldsymbol{g}_{\text{DICs}} \implies \begin{cases} \underbrace{S_{\text{s},1}\Delta\bar{g}_1}_{\text{LHS}} \approx \underbrace{\Delta g_{\text{s}} - S_{\text{s},2}\Delta\bar{g}_2}_{\text{RHS}} \\ S_{\text{u},2}\Delta\bar{g}_2 \approx \Delta g_{\text{u}} - S_{\text{u},1}\Delta\bar{g}_1 \end{cases}$$

By definition, the left-hand side (LHS) must have the same sign as the right-hand side (RHS) once compensation is applied. Compensation failure can occur when the RHS has a sign opposite to the sensitivity in the LHS. In other words, the compensation structure forces $\Delta\bar{g}_1$ to be negative. In such cases, whether compensation succeeds or fails depends on the initial value of $\bar{g}_1$ before compensation:

$$\bar{g}_{1,\text{after compensation}} = \bar{g}_{1,\text{before compensation}} + \Delta\bar{g}_1 \begin{cases} > 0 \longrightarrow \text{Compensation achieved} \\ < 0 \longrightarrow \text{Compensation failed} \end{cases}$$

Thus, compensation failure is stochastic and depends on the sampled conductance values. However, in practice, for a given target in the DICs space, compensation will most likely either succeed or fail with high probability. This is due to the problem different scales, which make the procedure almost insensitive to variance in the distribution.

5.3.2 A Numerical Procedure for Reachability Analysis

To identify a heuristic method for selecting which conductances to compensate, we propose a numerical procedure that constructs **reachability maps**. A reachability map is a visual representation of the likelihood that a target point in the DICs space can be reached through compensation, with areas of high reachability showing points where compensation is likely to succeed, and areas of low reachability showing points where compensation is unlikely. Using this procedure, we approximate the reachability map for each pair of compensated conductances in the second step of the generation: $\bar{\boldsymbol{g}}_{\text{comp.}} = (\bar{g}_i; \bar{g}_j)$, with $i, j \in \{\text{Na}, \text{Kd}, \text{CaT}, \text{CaS}, \text{KCa}, \text{A}, \text{H}\}$.

The procedure should ideally identify the two potential failure modes of compensation: those related to numerical instabilities and those related to the effect of opposite sign sensitivity. Algorithm 2 (Appendix C) outlines the step-by-step procedure we applied for each compensation pair.

To evaluate the coverage of the DICs space, we attempt to reach each target point on a predefined grid in the $(g_{\text{s}}, g_{\text{u}})$ space. For each point, we generate a population of instances and count how often the generation procedure successfully reaches that point — success is defined by matching the target DIC values within a specified tolerance.

Next, we calculate the *reachability percentage*, which corresponds to the proportion of the population that remains after filtering. This metric allows us to distinguish between points that are highly reachable — where the reachability percentage is close to 100% — and those that are largely unreachable, with values near 0%.

Figure 5.3 illustrates the results of this procedure for each pair of compensated conductances. We observe that reachability strongly depends on the choice of compensation set. Moreover, it appears that the entire DICs space is reachable, provided that the appropriate conductances are compensated. This is particularly relevant for our dataset generation process, as it confirms that with a well-chosen heuristic (a good choice of conductances), the generation method is highly effective. We can highlight a few observations:

- In Fyon et al., 2024, the authors examined the transition from tonic spiking to bursting, characterized by $g_{\text{s}} > 0$ and $g_{\text{s}} < 0$, respectively. They achieved this by compensating $\bar{g}_{\text{CaS}}$ and $\bar{g}_{\text{A}}$. Observing the corresponding reachability map, we see that this choice is particularly well-founded, as it allows for substantial control over g_{s}. In contrast, analyzing the impact on g_{u} would be more challenging, as its reachability is more constrained.

- As expected, compensating $\bar{g}_{\mathrm{Na}}$ and $\bar{g}_{\mathrm{Kd}}$ provides no control over the slow and ultra-slow dynamics. Furthermore, reachability maps where these conductances are compensated in combination with another conductance reveal much more restricted regions of reachability. This reinforces our argument that only conductances that significantly influence the target timescales should be selected for compensation.

- The reachability map for the compensation of $\bar{g}_{\mathrm{CaS}}$ and $\bar{g}_{\mathrm{CaT}}$ appears unusual, as very few points are reachable despite their expected impact on both slow and ultra-slow timescales. Further analysis is required to understand this phenomenon, but one possible explanation is that the iterative generation method fails to converge when these two calcium conductances are both part of the compensation set at the same time.

A Heuristic for Selecting Conductances to Compensate

Using the reachability maps from Figure 5.3, we can establish a rule for selecting the conductances to compensate, leading to a high probability of successful population generation. There are multiple possible choices, and the primary objective is to construct a combination of maps that ensures full exploration of the DICs space. The rule applied throughout this work is:

$$\bar{\boldsymbol{g}}_{\mathrm{comp.}} = \begin{cases} (\bar{g}_{\mathrm{CaS}}, \bar{g}_{\mathrm{H}}), & \text{if } g_{\mathrm{s}} < 0 \\ (\bar{g}_{\mathrm{A}}, \bar{g}_{\mathrm{H}}), & \text{if } g_{\mathrm{s}} > 0 \end{cases} \tag{5.6}$$

This specific choice is motivated by two arguments:

1. It involves only two distinct compensation sets and three variables in total, making it a simple yet effective rule for generating degenerate populations with diverse behaviors.

2. The choice of $\bar{g}_{\mathrm{CaS}}$ over $\bar{g}_{\mathrm{CaT}}$ is based on observations from Figure 2.17. Specifically, the sensitivity associated with $\bar{g}_{\mathrm{CaT}}$ is significantly lower than that of $\bar{g}_{\mathrm{CaS}}$ (Fig. 2.13). This lower sensitivity implies much larger variations in $\bar{g}_{\mathrm{CaT}}$ during compensation. Beyond the increased control cost, compensating $\bar{g}_{\mathrm{CaT}}$ sometimes leads to *one-spike bursting* behavior in the STG, likely due to the large variations imposed by the compensation process. Moreover, these choices are not devoid of biological plausibility. For instance, Temporal et al., 2012 show that the conductances $\bar{g}_{\mathrm{A}}$ and $\bar{g}_{\mathrm{H}}$ are co-regulated in motor neurons, ensuring stable neuronal function.

5.3.3 Reflections and Perspectives

The identification of the reachability problem in the DICs space, as well as the development of reachability maps as a response, provides a novel perspective on the DICs theory.

About biological plausibility A first consideration is that, although the entire DICs space appears to be theoretically explorable, this does not necessarily inform us about what's happening in biology. Two observations can be made: (1) The fact that a point in the DICs space is reachable does not guarantee that the corresponding activity is actually observable in biology; and (2) Our reachability analysis focuses solely on excluding negative conductance values, which we deem biophysically unrealistic. However, it is highly likely that biologically plausible compensations are also constrained by an upper bound on maximal conductance values through the number of channels that can be activated on the membrane.

Both of these aspects warrant further investigation using experimental data. The tools developed in this work could serve as an entry point for estimating DICs that are actually observed in biological systems.

About biological exploration These reachability maps could provide new insights into certain biological neuromodulation mechanisms. While the pipeline developed in this study allows us to understand how a perturbation influences neuronal activity through the lens of DICs, it does not directly reveal which conductances have been altered — since degeneracy implies that multiple solutions may exist. However, reachability maps could be used to determine which pairs of conductances allow transitions from one point in the DICs space (before perturbation) to another (after perturbation).

About more constrained compensation Another open question is the analysis of reachability when compensation is subject to more than two constraints (e.g., incorporating additional timescales or imposing additional constraints on the DIC curves). Does the DICs space remain unbounded in such cases? Do we observe disconnected regions in models with nonlinear compensation, and if so, does this disconnection manifest in biological systems as a safeguard separating different activity zones — requiring perturbations in additional ion channels to traverse?

In summary, although reachability analysis was introduced in this work as a technical solution, it may hold deeper implications and could be of interest for future research in neurosciences and the DICs theory.

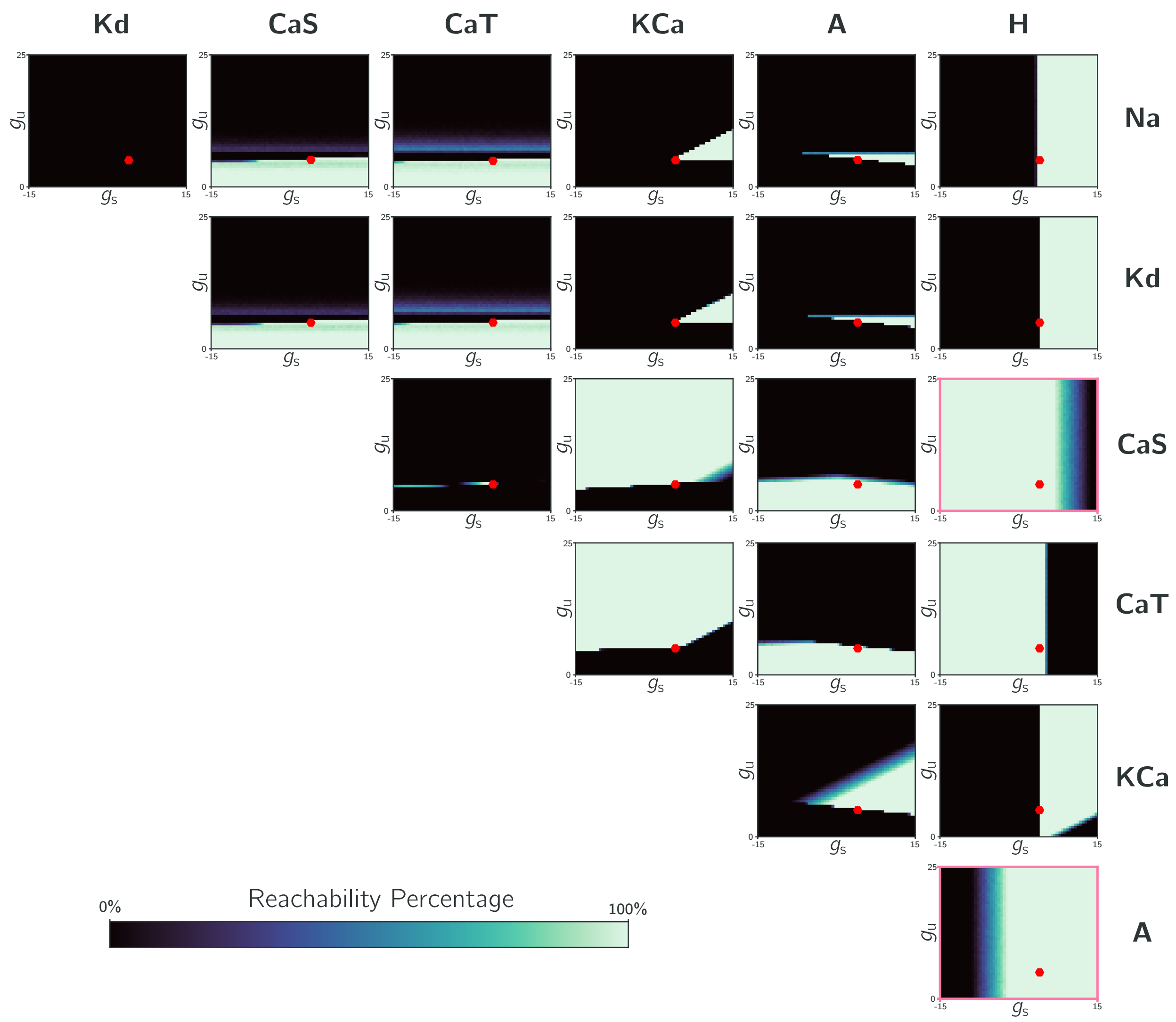


Figure 5.3: Reachability maps for different compensated conductance pairs in the STG model. Each heatmap represents the probability of reaching a given point in the DICs space when compensating a specific pair of conductances in the stomatogastric ganglion (STG) model. Dark regions indicate areas where compensation fails (low reachability), while light green regions correspond to high reachability. The red marker denotes the initial position of the population in the DICs space before the second step of the generation method. The two maps outlined in pink highlight the compensation pairs selected for the heuristic rule, demonstrating that their combination allows full coverage of the DICs space.

5.4 Dataset Generation

5.4.1 Procedure Overview

We generate the dataset necessary for training the deep learning architecture using our iterative generation procedure and relying on our heuristic for selecting conductances to compensate based on target DIC values. We selected $N = 75,000$ points in the DICs grid and generated a degenerate population of size $M = 16$ for each of these target values. Generating a complete population rather than a single individual helps the pipeline to generalize, as there is slight variability in activities within the same population. The resulting $|\mathcal{T}| = 1,200,000$ instances are split into a training set and a validation set of sizes $|\mathcal{T}_{\text{train}}| = 1,000,000$ and $|\mathcal{T}_{\text{val}}| = 200,000$, respectively. During this split, instances from the same population are not separated, since they share the same target values — mixing them would be equivalent to data leakage and could result in over-optimistic metrics when evaluating on the validation set. Each instance is then simulated, the spike time sequences are extracted, classified, and activity metrics are calculated (Appendix D). Silent instances are removed. Figure 5.4 illustrates the overall procedure.

We generate the test set, $\mathcal{T}_{\text{test}}$, in a similar manner. The test set has the same size as the validation set. To avoid any data leakage, it is generated well after the training and validation sets, once all choices related to the architecture and its training have been finalized. It represents 12,500 populations in the DICs space.

To pick points in the DICs space, we use *Latin Hypercube Sampling* (LHS) (McKay et al., 1979) within the range introduced in Appendix E. LHS is a method for selecting points in a multi-dimensional space to ensure they are spread out evenly. Instead of picking points randomly, which might leave some areas with too few or too many points, LHS divides the space into a grid and samples from each part of the grid. This way, the entire space is covered more uniformly, making it ideal for efficiently exploring complex spaces with fewer samples. For this dataset size, it has minimal impact and is equivalent to uniform sampling in the DICs space. However, this method will allow for better analysis when evaluating the impact of dataset size on generalization results, especially for smaller datasets, as it ensures better coverage of the DICs space.

5.4.2 Dataset Exploration

Here, we explore the training set as a preliminary step to training the pipeline. The goal is to provide a good understanding of the data and the various distributions observed.

Among all instances, 51.58% exhibit spiking activity, 48.28% exhibit bursting activity, and the remaining 0.24% are silent instances, which we discarded. Notably, the generation procedure imposing a strongly negative g_{f} is very effective in generating instances with spontaneous activity.

We analyze the *homogeneity* of populations based on the distribution of activity types within the same population. This analysis highlights regions in the DICs space where degeneracy is not effectively achieved, considering that the type of activity is central to neuronal function. We use entropy as a measure of homogeneity. An identical population has an entropy of 0, and a population with a uniform distribution of activity types — the most heterogeneous populations — has an entropy of 1. Formally, the entropy $\mathcal{H}(P)$ of a population P is defined as: $\mathcal{H}(P) = -\sum_{t \in \{\text{silent,spiking,bursting}\}} p_t \log_3(p_t)$, where p_t is the proportion of instances exhibiting activity t within the population. The base of the logarithm is chosen so that the entropy is correctly normalized between 0 (completely homogeneous) and 1 (completely heterogeneous). Figure 5.5 illustrates the distribution of entropy in the DICs space. There is clearly a zone of high heterogeneity when transitioning from negative g_{s} to positive g_{s}, which corresponds to the bursting/spiking transition. This observation is interesting because it implies that during training, the model may struggle with instances from this region, potentially generating outliers in the error distribution and affecting learning performance. This observation forms the basis for choosing a loss function that is less sensitive to outliers, as detailed in the following section. The second zone of high heterogeneity (in the bottom right of the DICs space) corresponds to populations containing silent instances.

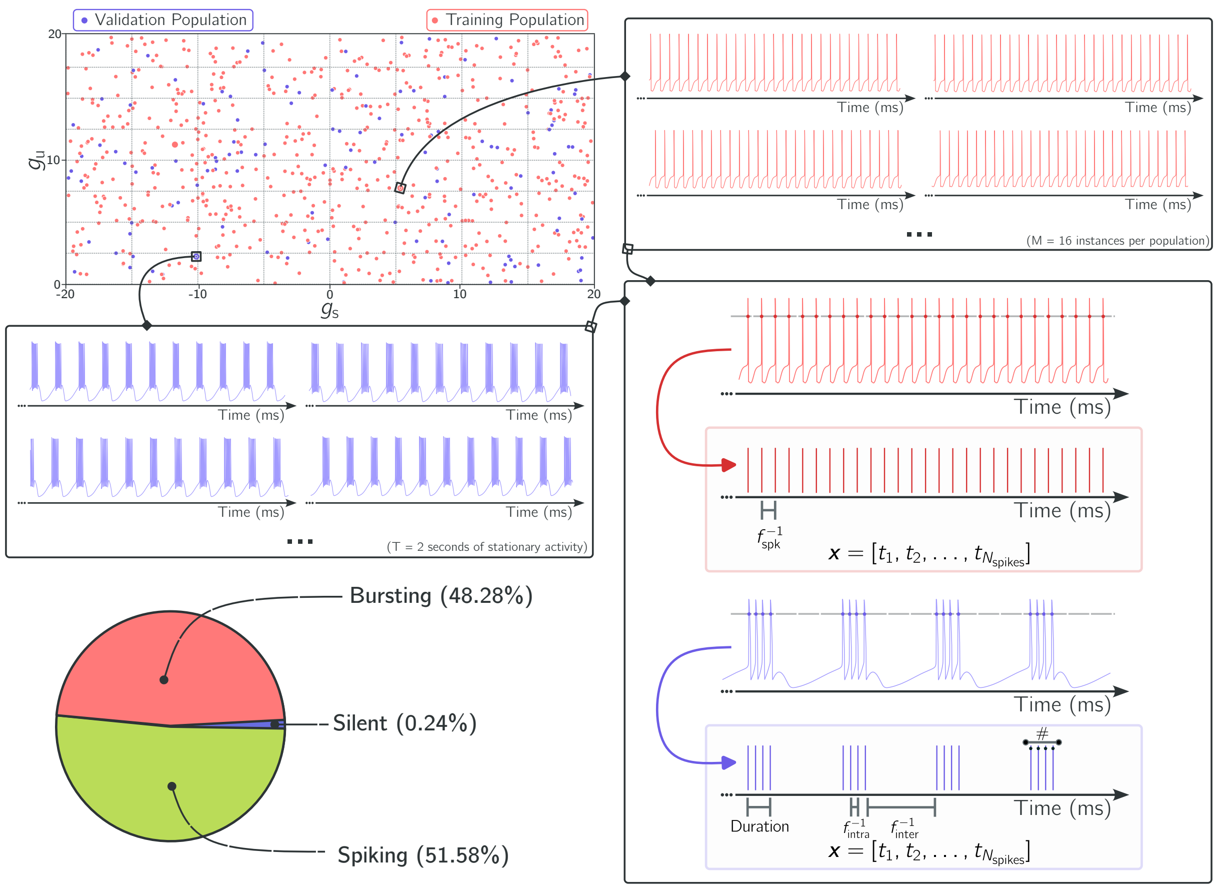


Figure 5.4: Overview of the dataset generation procedure. This figure illustrates the process of generating the dataset used for training the deep learning architecture. The procedure involves selecting $N = 75,000$ points in the DICs grid using Latin Hypercube Sampling and generating a degenerate population of size $M = 16$ for each target value. The resulting $1,200,000$ instances are split into training (red) and validation (purple) sets, with instances from the same population kept together. Each instance is simulated to extract spike time sequences, which are then classified and analyzed for activity metrics as defined in Appendix D.1. Silent instances are removed to ensure the dataset relevance for training. The bottom left pie chart shows the distribution of activity types, with 51.58% exhibiting spiking activity, 48.28% exhibiting bursting activity, and 0.24% being silent instances.

In addition to the type of activity, we can briefly study the various activity metrics presented in the Appendix D. The distribution of the different activity metrics is shown in Figure 5.6 and Table 6.2c reports the standard deviation for each of them. We observe that the STG is capable of presenting a wide variety of voltage activities. An in-depth analysis of these distributions is beyond the scope of this work, and we do not investigate further why certain features are bimodal while others are unimodal. Finally, we can examine the distribution of features in the DICs space. Figure 5.7 illustrates this distribution, and we observe that these descriptors of neuronal activity are not randomly distributed in the DICs space. This provides evidence that there are indeed relationships between the structure of the DICs space and neuronal activity — relationships that we aim for our pipeline to learn.

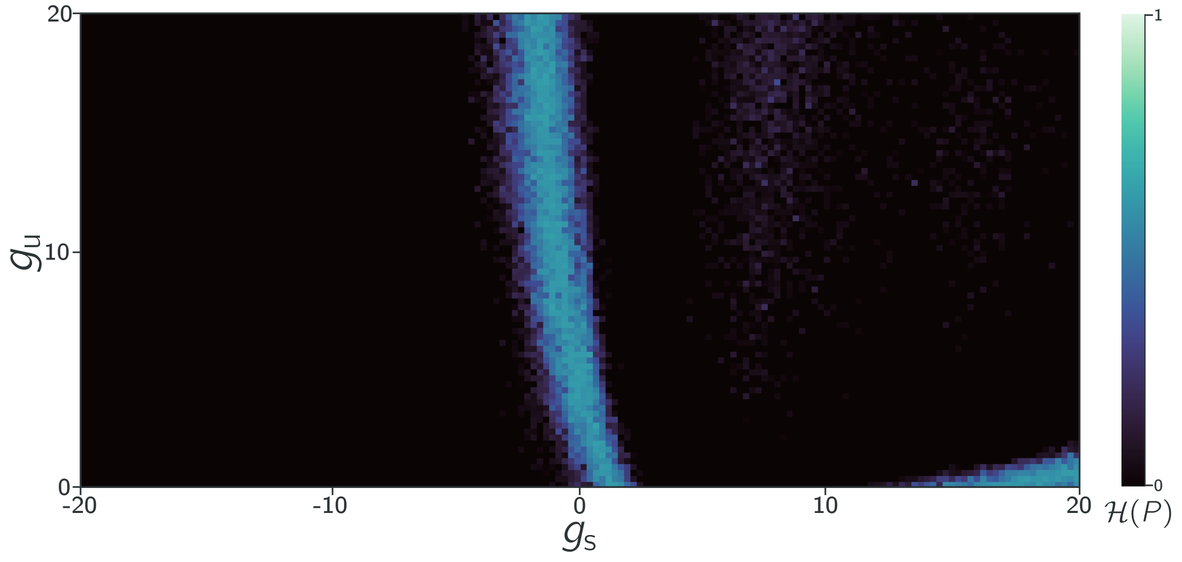


Figure 5.5: Homogeneity of populations in the DICs space. This figure shows the distribution of entropy across the DICs space, highlighting regions of high heterogeneity in neuronal activity types. Entropy is used as a measure of homogeneity, with 0 indicating completely homogeneous populations and 1 indicating maximally heterogeneous populations. The figure reveals two primary zones of high heterogeneity: one at the transition from negative to positive g_s, corresponding to the bursting/spiking transition, and another in the bottom right of the DICs space, where silent instances are present. These regions may pose challenges during training, as they can introduce outliers and affect model performance.

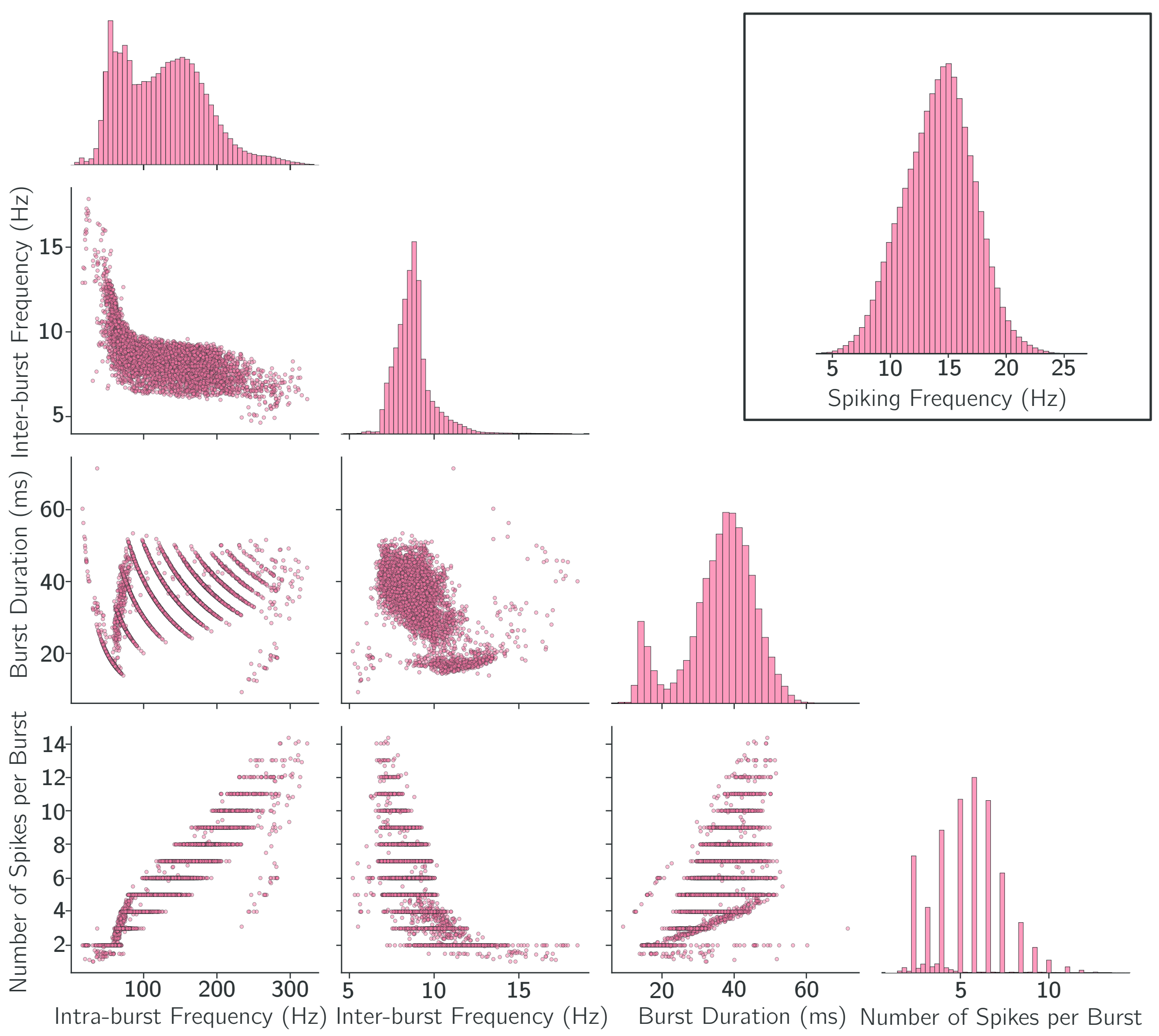


Figure 5.6: Distribution of activity metrics in the dataset. This figure illustrates the distributions of various activity metrics extracted from the dataset, including intra-burst frequency, inter-burst frequency, burst duration, the number of spikes per burst, and spiking frequency. The scatter plots display pairwise relationships between these metrics for a subset of the data (1%), while the histograms show the full distributions of each metric across the entire dataset. Notably, the top right panel highlights the distribution of spiking frequency. The STG model demonstrates a wide range of neuronal activities captured in the dataset.

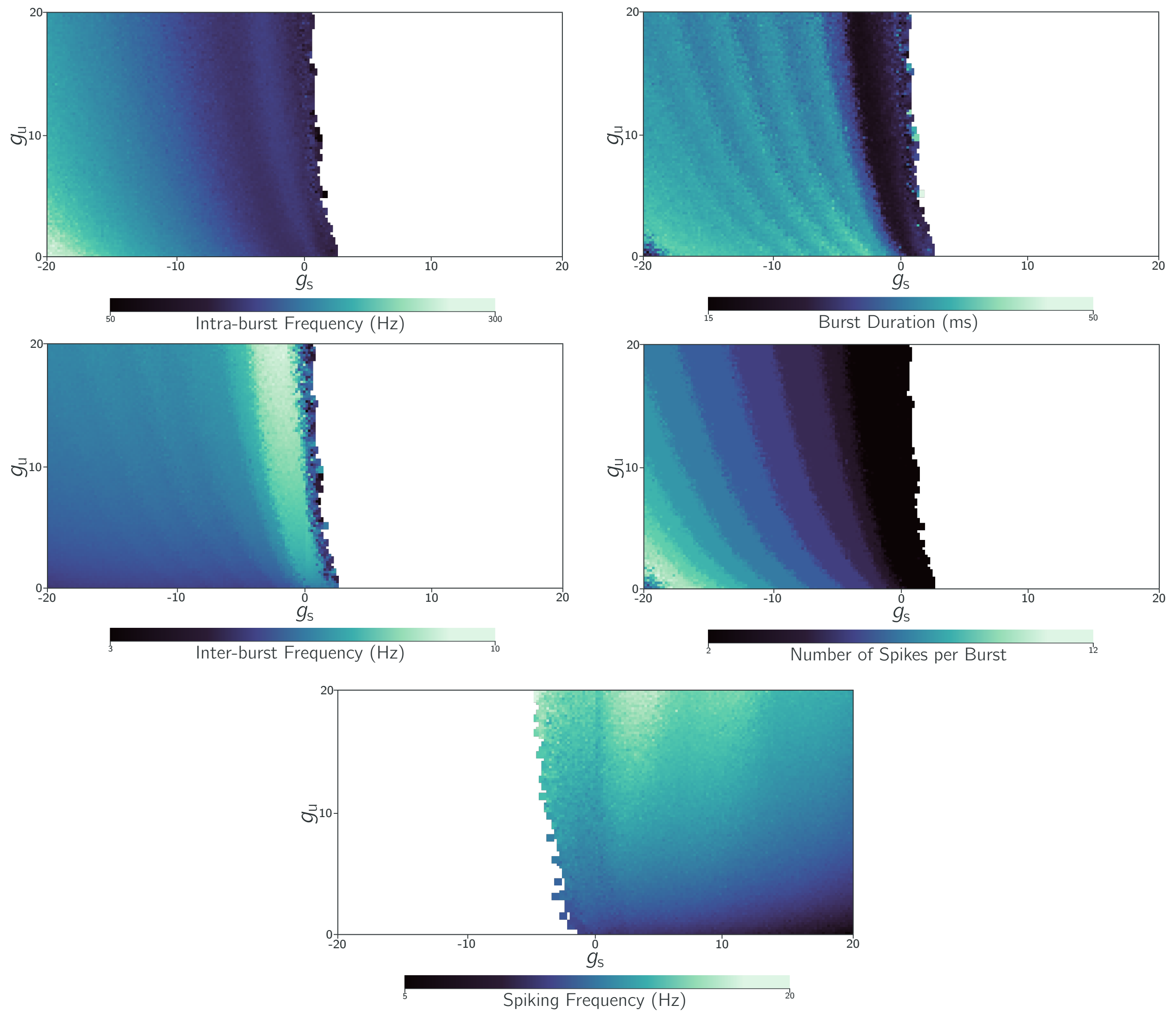


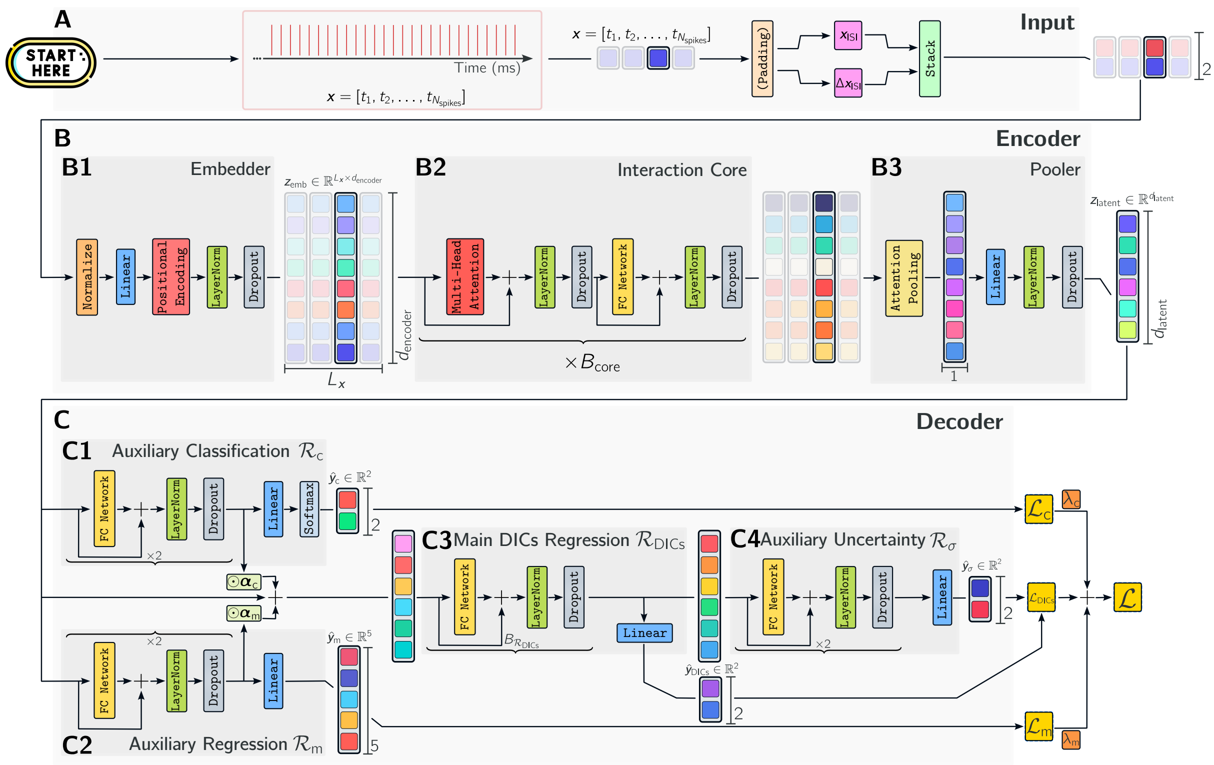
Figure 5.7: Distribution of activity metrics in the DICs space. This figure illustrates the distribution of various activity metrics across the DICs space, including intra-burst frequency, inter-burst frequency, burst duration, the number of spikes per burst, and spiking frequency. The four top panels focus on metrics related to bursting activity, while the bottom centered panel highlights spiking frequency. Each subplot represents the distribution of a specific metric as a function of the slow (g_s) and ultra-slow (g_u) conductances. The color gradients indicate the magnitude of each metric, with darker regions representing lower values and lighter regions representing higher values. The figures reveal structured patterns across the space, supporting the hypothesis that activity metrics are tightly linked to underlying DICs values. This visualization highlights the non-random distribution of neuronal activity descriptors in the DICs space, suggesting underlying relationships between the structure of the DICs space and neuronal activity patterns. These relationships are crucial for the pipeline to learn and generalize effectively.

5.5 Model Architecture and Training Strategy

This section details the exact architecture used in this thesis, as well as the training procedure. It describes the specific steps of this work following Box VI. The DNN is the part of the pipeline that takes as input the vector of spike times and output the corresponding target DIC values.

5.5.1 Building Our Deep Neural Networks

Using the various layers presented in Section 3.2.2, we construct the encoder and decoder of our pipeline.



A
START HERE
Time (ms)
$x = [t_1, t_2, \ldots, t_{N_{\text{spikes}}}]$
(Padding)
x_{ISI}
Δx_{ISI}
Stack
Input
2
B
Encoder
B1 Embedder
Normalize
Linear
Positional Encoding
LayerNorm
Dropout
$z_{\text{emb}} \in \mathbb{R}^{L_x \times d_{\text{encoder}}}$
d_{encoder}
L_x
B2 Interaction Core
Multi-Head Attention
FC Network
$\times B_{\text{core}}$
B3 Pooler
Attention Pooling
$z_{\text{latent}} \in \mathbb{R}^{d_{\text{latent}}}$
d_{latent}
1
C
Decoder
C1 Auxiliary Classification $\mathcal{R}_c$
Softmax
$\hat{y}_c \in \mathbb{R}^2$
×2
$\odot \alpha_c$
$\odot \alpha_m$
C2 Auxiliary Regression $\mathcal{R}_m$
$\hat{y}_m \in \mathbb{R}^5$
5
C3 Main DICs Regression $\mathcal{R}_{\text{DICs}}$
$B_{\mathcal{R}_{\text{DICs}}}$
$\hat{y}_{\text{DICs}} \in \mathbb{R}^2$
C4 Auxiliary Uncertainty $\mathcal{R}_\sigma$
$\hat{y}_\sigma \in \mathbb{R}^2$
$\mathcal{L}_c$
λ_c
$\mathcal{L}_{\text{DICs}}$
$\mathcal{L}$
$\mathcal{L}_m$
λ_m

Figure 5.8: Model architecture and training strategy. This figure illustrates the architecture and training strategy of the deep neural network used in this thesis. (A) The input to the model consists of spike time sequences, from which inter-spike intervals (ISIs) and delta ISIs are extracted. These features are then stacked and fed into the encoder. (B) The encoder processes the input through three main components: the embedder, the interaction core, and the pooler. The embedder transforms the input sequence into a normalized, higher-dimensional representation. The interaction core processes this representation using multi-head attention mechanisms and fully-connected networks. The pooler aggregates the variable-length representation into a fixed-size latent representation. (C) The decoder transforms the fixed-size latent representation into the DICs space, $\mathbb{R}^2$. It includes auxiliary tasks for classifying neuronal activity and regressing electrical activity metrics, as well as predicting uncertainty measures. The training strategy incorporates dataset augmentation, auxiliary tasks, and a robust loss function to enhance model performance and generalization.

The Encoder

The architecture of the encoder can be conveniently described in three parts (Figure 5.8B). The first is the **embedder** (B1), which aims to transform the input sequence of spike times into a normalized, higher-dimensional representation that can be practically used by the rest of the encoder. The second is the **interaction core** (B2), which processes the input using attention mechanisms. Finally, the encoder concludes with the **pooler** (B3), which transforms the variable-length latent representation into a fixed-size representation.

The embedder receives the sequence of spike times as input. When the pipeline is used in batch mode, i.e., using a single forward pass for multiple sequences simultaneously, these sequences are first transformed via a padding layer. Padding masks are generated and passed to the various blocks of the pipeline that require them. Each sequence then undergoes a series of transformations aimed at better processing by the attention core.

Specifically, since electrical activity is time-translation invariant (meaning that the activity is described by the relative values separating spike times rather than their absolute values), the sequence in absolute timing is transformed into an inter-spike intervals (ISIs) sequence: $\boldsymbol{x} \to \boldsymbol{x}_{\text{ISI}} = \Delta\boldsymbol{x}$ (Appendix D). We propose augmenting the input representation with the vector $\Delta\boldsymbol{x}_{\text{ISI}}$, which encodes the variation observed between two successive ISIs. We suggest that this value allows the rest of the pipeline to more easily identify ISIs within a burst (with a small ΔISI value) and ISIs separating bursts (with a large ΔISI value). We thus construct (Fig. 5.8A):

$$\boldsymbol{x}_{\text{features}} = \frac{\begin{bmatrix} x_{\text{ISI, 1}} & x_{\text{ISI, 2}} & \cdots \\ \Delta x_{\text{ISI, 1}} & \Delta x_{\text{ISI, 2}} & \cdots \end{bmatrix} - \boldsymbol{\mu}_{\text{train}}}{\boldsymbol{\sigma}_{\text{train}}} \in \mathbb{R}^{*\times 2},$$

where $\boldsymbol{\mu}_{\text{train}}$ and $\boldsymbol{\sigma}_{\text{train}}$ are fixed vectors that normalize the representation. Their values are calculated on the training set as the mean (of the ISI values and the ΔISI values) and their standard deviations, respectively.

Next, the sequence is projected into a space $\mathbb{R}^{*\times d_{\text{encoder}}}$ via a linear layer. It is important to note that throughout the pipeline, only the attention layers and the pooling layer interact with different elements of the sequence together. In other words, when applying a linear layer or a normalization layer to a sequence, it is applied element-wise. Finally, the embedder introduces positional encoding:

$$\boldsymbol{z}_{\text{emb}} = \boldsymbol{W}\boldsymbol{x}_{\text{features}} + \boldsymbol{P} \in \mathbb{R}^{*\times d_{\text{encoder}}}.$$

Optionally — and this is studied during hyperparameter tuning — a logarithmic transformation can be applied to $\boldsymbol{x}_{\text{ISI}}$. This operation transforms the distribution of ISIs to give more room to differences in small values and to crush differences between large values. In certain situations, this operation can stabilize training and increase convergence.

$$\boldsymbol{x}_{\text{ISI}} \xrightarrow{\text{Should log transform ?}} \boldsymbol{x}_{\text{ISI}} \leftarrow \log\left(1 + \boldsymbol{x}_{\text{ISI}}\right)$$

The interaction core is the central component of the encoder, designed to process the embedded sequence using attention mechanisms. This core is structured to capture complex dependencies and interactions between different elements of the sequence.

The interaction core consists of multiple stacked layers, each containing a multi-head attention mechanism followed by a fully-connected network[1]. These layers are arranged sequentially to process the input sequence and capture hierarchical features.

To stabilize training and facilitate gradient flow, residual connections are used around each sub-layer (multi-head attention and fully-connected networks). Additionally, layer normalization is applied to the output of each sub-layer to maintain consistent scaling.

To prevent overfitting and improve generalization, dropout is incorporated into the interaction core. Dropout is applied after each sub-layer (multi-head attention and fully-connected networks), just after the residual connection and layer normalization.

[1] In this work, we call a *fully-connected network* a sequence of *fully-connected layers* interleaved with non-linear activation functions (e.g., ReLU, GELU) and *dropout* for regularization. Specifically, we use two linear layers with an activation function and dropout in between. This corresponds to the classical *multi-layer perceptron* (MLP) architecture with a single hidden layer.

The interaction core is composed of a sequence of B_{core} identical blocks. Each block contains a series of sub-layers, including multi-head attention, dropout, residual connections, layer normalization, and position-wise fully-connected networks. The detailed arrangement of these sub-layers is illustrated in Figure 5.8. This structure allows the interaction core to effectively process the input sequence, capturing both local and global dependencies, and preparing the data for the subsequent pooling layer.

The pooler is the final component of the encoder, transforming the variable-length latent representation from the interaction core into a fixed-size representation. This is crucial for bridging the encoder and decoder, ensuring a consistent input size for the decoder. The pooler uses self-attention pooling to aggregate information, reducing dimensionality while retaining essential features from, going from $\mathbb{R}^{*\times d_{\text{encoder}}}$ to $\mathbb{R}^{d_{\text{encoder}}}$.

Finally, a linear layer is used to enforce the latent space to be $\mathbb{R}^{d_{\text{latent}}}$. This transformation can be expressed as:

$$z_{\text{latent}} = \boldsymbol{W}_{\text{pool}} \boldsymbol{z} + \boldsymbol{b}_{\text{pool}},$$

where $\boldsymbol{W}_{\text{pool}} \in \mathbb{R}^{d_{\text{latent}} \times d_{\text{encoder}}}$ is the weight matrix and $\boldsymbol{b}_{\text{pool}} \in \mathbb{R}^{d_{\text{latent}}}$ is the bias vector. This transformation prepares the data for subsequent processing by the decoder.

The Decoder

The decoder is responsible for transforming the fixed-size representation produced by the encoder into the DICs space, $\mathbb{R}^2$ (Fig. 5.8C). It leverages the information processed by the encoder to generate meaningful outputs. The decoder consists of several components, each playing a different role in the transformation process:

Residual fully-connected networks are stacked sequentially to process the fixed-size representation from the encoder. Each residual FC network includes FC layers, residual connections, layer normalization, and dropout. These components work together to capture complex patterns in the data while stabilizing training and preventing overfitting.

The decoder contains $B_{\mathcal{R}_{\text{DICs}}}$ of these residual fully-connected networks, where each network processes the input sequentially. The detailed arrangement of these components is illustrated in Figure 5.8C3.

Final linear layer After the sequence of residual fully-connected networks, the decoder concludes with a final linear layer. This layer maps the processed representation into the DICs space, $\mathbb{R}^2$. The transformation can be mathematically represented as:

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_{\text{final}} \boldsymbol{z} + \boldsymbol{b}_{\text{final}},$$

where $\boldsymbol{z} \in \mathbb{R}^{d_{\text{latent}}}$ is the output from the last residual fully-connected network, $\boldsymbol{W}_{\text{final}} \in \mathbb{R}^{2\times d_{\text{latent}}}$ is the weight matrix, and $\boldsymbol{b}_{\text{latent}} \in \mathbb{R}^2$ is the bias vector.

5.5.2 Training Our Deep Neural Networks

This subsection outlines the strategies employed to train our DNN. It details the techniques used to enhance the model robustness and generalization capabilities, including dataset augmentation, the introduction of auxiliary tasks, and the design of a robust loss function. These strategies collectively aim to improve the model performance in regressing DIC values and ensure effective learning from spike time sequences.

Dataset augmentation To enhance the robustness and generalization of our model, we employ dataset augmentation techniques. Two methods are combined: (1) To make the spiking time measurements more similar to what can be recorded in a laboratory setting, we introduce noise into the data. Specifically, we add Gaussian noise to the spike times and apply dropout to randomly remove some spikes. (2) We use a subset of the complete sequence obtained from simulations. To do this, we sample the duration of the recording window and extract this window from the original sequence. This augmentation step is performed independently and anew each time a sequence is drawn from the training set. Given a sequence of spike times $\boldsymbol{x} = [t_1, t_2, \dots, t_{N_\text{spikes}}]$, we augment the data by:

1. Extracting a time window:

$$\boldsymbol{x}_\text{window} = \{t_i \mid t_i \in \boldsymbol{x},\ t_\text{start} \le t_i \le t_\text{end}\},$$

   where t_start is a randomly chosen start time within the interval $[0, T - D]$, $t_\text{end} = t_\text{start} + D$, D is the duration of the window sampled uniformly from $[T/2, T]$.

2. Adding Gaussian noise:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}_\text{window} + \boldsymbol{\epsilon}, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2_\text{noise}),$$

   where $\boldsymbol{\epsilon}$ is the noise vector sampled from a Gaussian distribution with mean 0 and variance σ^2_noise. We use $\sigma_\text{noise} = 2\,\text{ms}$.

3. Applying dropout:

$$\boldsymbol{x}_\text{aug} = \tilde{\boldsymbol{x}} \odot \boldsymbol{m}, \quad m_i \sim \text{Bernoulli}(1 - p),$$

   where $\boldsymbol{m}$ is a dropout mask vector sampled from a Bernoulli distribution with probability $1 - p$ of keeping each element, and $\odot$ denotes element-wise multiplication. We use $p = 5\%$.

This augmentation process helps the model generalize better by exposing it to variations in the input data that mimic real-world conditions. An illustration of this process on a sample sequence is provided in Figure 5.9.

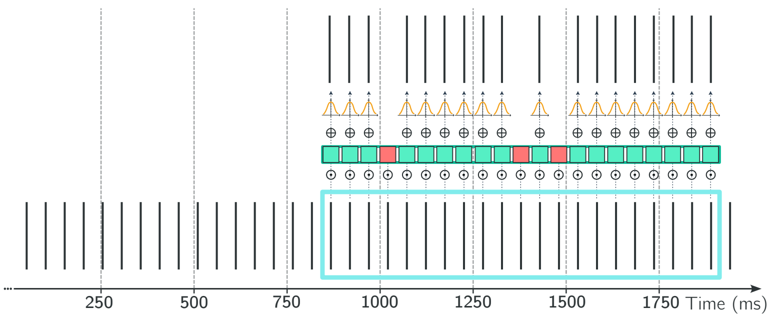


Figure 5.9: Illustration of data augmentation applied to spike time sequences. This figure visualizes the data augmentation process applied to spike time sequences during training. Starting from the original sequence (bottom row), a time window of variable duration is randomly selected to simulate partial recordings (cyan box). Dropout is then applied to randomly remove a subset of spikes (red blocks), mimicking imperfect spike detection. Gaussian noise is subsequently added to the remaining spike times (orange curves), modeling measurement noise. The final augmented sequence (top row) is used as input to the model. This augmentation strategy enhances robustness and generalization by introducing biologically plausible variability into the training data.

Auxiliary tasks Although the primary objective of the pipeline is to regress DIC values, we introduce two auxiliary tasks to assist the model during training. Specifically, these tasks are designed to help the encoder construct structured latent representations. Practically, we introduce two auxiliary heads, $\mathcal{R}_\text{m}$ (Fig. 5.8C2) and $\mathcal{R}_\text{c}$ (C1). The goal of $\mathcal{R}_\text{m}$ is to regress the metrics of electrical activity (Appendix D), while $\mathcal{R}_\text{c}$ aims to classify neural activity into spiking and bursting.

The head $\mathcal{R}_\text{c}$ consists of a sequence of two residual fully-connected networks with dropout and layer normalization. Its output $\hat{\boldsymbol{y}}_\text{c} \in \mathbb{R}^2$ represents the probabilities that the input sequence corresponds to spiking activity ($i = 1$) or bursting activity ($i = 2$).

The head $\mathcal{R}_\text{m}$ has a structure similar to the classification head. Its output $\hat{\boldsymbol{y}}_\text{m} \in \mathbb{R}^5$ provides the pipeline predictions for the following metrics: spiking frequency ($i = 1$), burst duration ($i = 2$), intra-burst frequency ($i = 3$), inter-burst frequency ($i = 4$), and the number of spikes per burst ($i = 5$).

The input to the two auxiliary heads is the latent representation z_latent produced by the encoder. Note that this approach is not in contradiction with the idea of avoiding the use of summary statistics as pipeline inputs. Rather, the aim is to help structuring the latent representation. Ultimately, the encoder is free to extract all the features it learns necessary. We simply guide the training process on the assumption that the encoder should summarize the neuronal activity in the latent space.

The last latent representations of the two auxiliary heads are then mixed — via a learnable averaging procedure, parametrized by $\boldsymbol{\alpha}_\text{c}, \boldsymbol{\alpha}_\text{m} \in \mathbb{R}^{d_\text{latent}}$ — with the latent representation of the encoder, the result serving as input to the main DICs regression head $\mathcal{R}_\text{DICs}$ (Fig. 5.8C3):

$$x_{\mathcal{R}_\text{DICs}} = z_\text{latent} + \boldsymbol{\alpha}_\text{m} \odot z_\text{m} + \boldsymbol{\alpha}_\text{c} \odot z_\text{c}$$

Additionally, we introduce an auxiliary head $\mathcal{R}_\sigma$ to predict the uncertainty measure for the DICs (C4). This head outputs one value of uncertainty per output, specifically σ_{g_s} and σ_{g_u}. It consists of two residual fully-connected networks with dropout and layer normalization. The input to $\mathcal{R}_\sigma$ is the last latent representation of $\mathcal{R}_\text{DICs}$ before the final linear layer.

Loss function The loss function used in our training procedure is designed to handle outliers and uncertainty measures. Specifically, for the primary task of regressing DIC values, we employ the heteroscedastic Huber loss function. This choice is motivated by the heterogeneity exploration that has revealed the possible existence of outliers, which could make training less effective if a standard mean squared error (MSE) loss were used (Section 5.4.2). The heteroscedastic Huber loss is less sensitive to outliers and also accounts for the varying uncertainty in predictions, providing a more robust training process.

The total loss $\mathcal{L}$ is a weighted sum of the primary task loss $\mathcal{L}_\text{DICs}$ and the auxiliary task losses $\mathcal{L}_\text{m}$ and $\mathcal{L}_\text{c}$:

$$\mathcal{L} = \mathcal{L}_\text{DICs} + \lambda_\text{m}\mathcal{L}_\text{m} + \lambda_\text{c}\mathcal{L}_\text{c},$$

where λ_m and λ_c are hyperparameters that control the contribution of the auxiliary tasks to the total loss.

For the primary task of regressing DIC values, the loss function $\mathcal{L}_\text{DICs}$ is defined as:

$$\mathcal{L}_\text{DICs} = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{\sigma^2_{g_\text{s},i}}\text{Huber}(y_{g_\text{s},i} - \hat{y}_{g_\text{s},i}) + \log(\sigma^2_{g_\text{s},i}) + \frac{1}{\sigma^2_{g_\text{u},i}}\text{Huber}(y_{g_\text{u},i} - \hat{y}_{g_\text{u},i}) + \log(\sigma^2_{g_\text{u},i})\right],$$

where $y_{g_\text{s},i}$ and $y_{g_\text{u},i}$ are the ground truth values for the DICs, $\hat{y}_{g_\text{s},i}$ and $\hat{y}_{g_\text{u},i}$ are the predicted values, N is the number of samples, $\sigma^2_{g_\text{s},i}$ and $\sigma^2_{g_\text{u},i}$ are the predicted uncertainties for the i-th sample, and Huber$(\cdot)$ is the Huber loss function.[2] The Huber loss is defined as:

$$\text{Huber}(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \le \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases},$$

where δ is a threshold parameter that determines the point at which the loss function transitions from quadratic to linear. We use $\delta = 1$.

For the auxiliary task of regressing electrical activity metrics, the loss function $\mathcal{L}_\text{m}$ is defined as:

$$\mathcal{L}_\text{m} = \frac{1}{N}\sum_{i=1}^{N} \|\,(\boldsymbol{y}_{\text{m},i} - \hat{\boldsymbol{y}}_{\text{m},i}) \odot \boldsymbol{m}_i\|^2,$$

[2]Note that although the same symbol is used, the uncertainty measure are not strictly the variances associated with classical distributions — unlike if a heteroscedastic mean squared error (MSE) had been used.

where $\boldsymbol{y}_{\mathrm{m},i}$ is the ground truth for the electrical activity metrics, $\hat{\boldsymbol{y}}_{\mathrm{m},i}$ is the predicted value, and $\boldsymbol{m}_i$ is a masking vector that selects the relevant metrics based on the type of activity. The masking vector $\boldsymbol{m}_i$ is defined as:

$$\boldsymbol{m}_i = \begin{cases} [1,0,0,0,0] & \text{if } y_{\mathrm{c},i} = 1 \text{ (spiking activity)} \\ [0,1,1,1,1] & \text{if } y_{\mathrm{c},i} = 2 \text{ (bursting activity)} \end{cases},$$

where $y_{\mathrm{c},i}$ is the ground truth label indicating whether the i-th sequence corresponds to spiking or bursting activity.

This loss function ensures that only the relevant metrics are considered based on the type of activity. Specifically, for spiking sequences ($y_{\mathrm{c},i} = 1$), only the spiking frequency metric is included in the loss calculation. For bursting sequences ($y_{\mathrm{c},i} = 2$), only the burst duration, intra-burst frequency, inter-burst frequency, and number of spikes per burst metrics are included.

For the auxiliary task of classifying neuronal activity, the loss function $\mathcal{L}_\mathrm{c}$ is defined as:

$$\mathcal{L}_\mathrm{c} = -\frac{1}{N}\sum_{i=1}^{N}\left[\mathbb{I}(y_{\mathrm{c},i}=1)\log(\hat{y}_{\mathrm{c},i,1}) + \mathbb{I}(y_{\mathrm{c},i}=2)\log(\hat{y}_{\mathrm{c},i,2})\right],$$

where $\hat{y}_{\mathrm{c},i,1}$ and $\hat{y}_{\mathrm{c},i,2}$ are the predicted probabilities for spiking activity and bursting activity, respectively, for the i-th sample.[3]

In summary, our training strategy incorporates dataset augmentation, auxiliary tasks, and a robust loss function to enhance the model performance and generalization. The auxiliary tasks, $\mathcal{R}_\mathrm{m}$ and $\mathcal{R}_\mathrm{c}$, help structure the latent representation by regressing activity metrics and classifying activity, respectively. Additionally, the auxiliary head $\mathcal{R}_\sigma$ predicts the uncertainty measures for the DICs, further improving the model robustness. The overall training pipeline, including the auxiliary tasks and their associated losses, is illustrated in Figure 5.8.

Evaluation metrics During evaluation, two primary metrics are used. The loss $\mathcal{L}_\mathrm{DICs}$ is employed to assess whether the model is learning correctly and is also used for hyperparameter tuning on the validation set $\mathcal{T}_\mathrm{val}$. For the final evaluation on the test set $\mathcal{T}_\mathrm{test}$, an additional performance measure is introduced. The reasoning is that interpreting the results of DICs regression directly is challenging, as it is difficult to translate the regression error into a meaningful interpretation of the quality of the generated populations. Therefore, we use the van Rossum distance, introduced in the preliminary elements (Appendix D.2). Since this distance measures the similarity between two activities, we extend its definition to function as follows.

The goal is to determine if the pipeline can generate, based on a single sequence $\boldsymbol{x} \in P$, a population P' whose overall activity is similar to that of $\boldsymbol{x}$. We calculate the average van Rossum distance between $\boldsymbol{x}$ and the other instances in the population P:

$$d_\mathrm{VR}^{P}(\boldsymbol{x}) = \frac{1}{|P|-1}\sum_{\boldsymbol{x}' \in P, \boldsymbol{x} \neq \boldsymbol{x}'} d_\mathrm{VR}(\boldsymbol{x}, \boldsymbol{x}'),$$

and we calculate the average van Rossum distance between $\boldsymbol{x}$ and the other instances in P', denoted as $d_\mathrm{VR}^{P'}(\boldsymbol{x})$. For these calculations, we use the midrange ISI value of $\boldsymbol{x}$ as the value of τ in the van Rossum distance. This gives us a data-driven approach to adapt the hyperparameter of the distance, assuming that the timescale that matters is the one of the baseline activity. Finally, we use the ratio of these distances as the evaluation metric:

$$R_\mathrm{VR}(\boldsymbol{x}) = \frac{d_\mathrm{VR}^{P'}(\boldsymbol{x})}{d_\mathrm{VR}^{P}(\boldsymbol{x})}.$$

[3]In cases where the dataset is unbalanced, techniques such as class weighting or resampling can be employed to ensure that the model does not become biased towards the majority class. Based on the dataset exploration, classes are relatively well balanced in the case of the STG model but not in the case of the DA model (Appendix G).

The interpretation is that if the pipeline is performing well, this ratio should be close to or less than 1. The denominator accounts for the inherent variability of the generation procedure. It is possible to obtain values of $R_{\mathrm{VR}}(\boldsymbol{x}) < 1$ since the pipeline is conditioned solely on $\boldsymbol{x}$ and not on the entire population P.

Additionally, we will study the robustness of the pipeline to noise in the input data. Specifically, we will investigate the impact of varying the intensity of input noise, σ_{noise} on the DICs loss measured on the test set given the trained pipeline. This analysis will help us understand to what extent the model generalizes in the presence of noisy data, which is particularly important for laboratory use on data recorded *in vitro*.

5.5.3 Exploring the Hyperparameters of Our Deep Neural Network

To achieve optimal performance from our pipeline, we explore various hyperparameters to identify a suitable architecture and training algorithm. We employ an extended random search procedure over $N_{\text{search}} = 100$ sets of hyperparameters. Table 5.2 summarizes the explored hyperparameters and their respective distributions. We train the model for $N_{\text{epoch}} = 50$ epochs — that is, the number of times the model is exposed to the training set $\mathcal{T}_{\text{train}}$. We evaluate the model on the validation set $\mathcal{T}_{\text{val}}$ using the DICs regression loss, $\mathcal{L}_{\text{DICs}}$, every quarter of an epoch. As training is quite time-consuming, only half of the training set is used for this hyperparameters tuning step; the size of the validation set remains unchanged. For each set of hyperparameters, the best performance obtained on the validation set is retained for comparison.

Hyperparameter	**Type / Distribution**	**Values or Range**
Learning rate (η)	Log-uniform	$[10^{-5}, 5 \cdot 10^{-3}]$
Dropout (p_{dropout})	Uniform	$[0.0, 0.4]$
Latent space dimension (d_{latent})	Discrete	$\{16, 32, 64, 128\}$
Encoder space dimension (d_{encoder})	Discrete	$\{16, 32, 64, 128\}$
Number of heads (H)	Discrete	$\{2, 4, 8\}$
Number of encoder blocks (B_{core})	Discrete	$\{1, 2, 3, 4, 5, 6, 7, 8\}$
Number of decoder blocks ($B_{\mathcal{R}_{\text{DICs}}}$)	Discrete	$\{2, 4, 6, 8, 10\}$
Activation function ($\sigma(\cdot)$)	Categorical	`relu`, `gelu`, `silu`, `tanh`
Apply log transform to input?	Boolean	`true`, `false`
Regression weighting factor ($\tilde{\lambda}_{\text{m}}$)	Uniform	$[0.01, 10.0]$
Classification weighting factor ($\tilde{\lambda}_{\text{c}}$)	Uniform	$[0.01, 10.0]$
Batch size ($\lvert B \rvert$)	Discrete	$\{16, 32, 64, 128\}$

Table 5.2: Hyperparameters explored during the extended random search and their distributions. The factors $\tilde{\lambda}_{\text{c}}$ and $\tilde{\lambda}_{\text{m}}$ represent the relative weights of the auxiliary losses with respect to the primary DICs loss. Specifically, if $\tilde{\lambda}_{\text{c}} = 10$, then the value of λ_{c} is calculated on the first batch of training such that the classification loss is 10 times more significant than the primary loss. "*Apply log transform to input?*" refers to whether the logarithmic transformation is applied (or not) to the sequence of ISIs.

In addition to identifying a suitable architecture for our task, the hyperparameter tuning procedure allows us to quantify the importance and impact of different hyperparameters on performance. This analysis provides a better interpretation of the task complexity and insights into potential avenues for improving the architecture. We will briefly analyze the general results of the hyperparameter exploration in Section 6.1.

5.5.4 Exploring the Data Requirements of Our Deep Neural Network

To evaluate the minimum amount of data required to effectively train our pipeline, we conduct a study on data efficiency. Specifically, we train the model on different fractions of the training set $\mathcal{T}_{\text{train}}$, while keeping the validation data fixed. The fractions considered are $f \in \{1\%, 5\%, 10\%, 25\%, 50\%, 75\%, 100\%\}$. To ensure good coverage of the DICs space after sub-sampling, we generate a new *Latin Hypercube Sampling* of the desired size and select the closest corresponding points from the original training set. We use the best hyperparameters obtained from the previous search. Since training convergence can depend on the size of the dataset, we increase $N_{\text{epoch}} = 100$ epochs.

This study has two objectives: (1) to compare our method and the gains achieved through DICs theory with the methods of Gonçalves et al., 2020; and (2) to optimize the transfer of the model to the dopaminergic (DA) neuron by leveraging the results obtained from the STG model, which benefits from the pretrained backbone on the STG model.

For each dataset fraction, performance is measured using $\mathcal{L}_{\text{DICs}}$ on the test set $\mathcal{T}_{\text{test}}$. The objective is to observe the performance curve as a function of the available data and to identify the threshold beyond which further improvements become marginal.

5.6 Transfer to the Dopaminergic Neuron Model

Using the same architecture and data generation procedure as for the STG model, and considering the dataset size identified from the previous procedure, we introduce LoRA adapters into the architecture trained on the STG model. We explore the optimal size of the low-rank matrices, r, using grid-search. The evaluation and validation metrics are the same as those used for the STG model. In addition, we will compare the results of the best adapter architecture with the results obtained by training a new architecture from scratch. The aim is to compare whether performance is indeed similar and whether the adapters method is a good solution for scaling our method to many CBMs in a single solution.

Appendix G shows the exact methodology used for the transfer to the DA model.

Chapter 6

Results and Discussion

This chapter presents the results related to the central question of this work: *can a Deep Neural Network combined with the theory of Dynamic Input Conductances be used to generate degenerate populations of Conductance-Based Models with target activity based on experimental measurements?* It should be noted that the discussions on reachability (Section 5.3) and the iterative compensation algorithm (5.2) are covered in their respective sections earlier in the thesis. Therefore, they are not revisited here but are implicitly foundational to the results presented in this chapter.

The chapter is organized to first discuss the results of hyperparameter tuning and data efficiency analysis, which are crucial for understanding the architecture performance and training requirements. Following this, the chapter focuses on the final model evaluation, including robustness analysis against input noise, and explores the effectiveness of transfer learning techniques applied to the dopaminergic neuron model.

By the end of this chapter, readers should have a comprehensive understanding of the effectiveness of the proposed methodology and its potential for application in generating neuronal populations with specific activity patterns.

Structure of the Chapter

The first section, **6.1 Hyperparameter Tuning and Data Requirements**, focuses on the results related to the architecture itself, detailing the outcomes of the hyperparameter tuning phase and the data requirements for training the STG model.

The second section, **6.2 Final Model Evaluation and Robustness Analysis**, presents the final performance of the architecture trained with the best hyperparameters, evaluates the quality of predictions on the test data, and analyzes the model robustness against various levels of noise introduced into the input data.

The third section, **6.3 Results of Transfer to the Dopaminergic Neuron Model**, discusses the experimental results obtained from adapting the architecture to the dopaminergic neuron model using LoRA adapters, demonstrating the effectiveness of this technique for extending the pipeline at a lower cost.

6.1 Hyperparameter Tuning and Data Requirements

This section focuses on the results related to the architecture itself. We detail the results of the hyperparameter tuning phase and the data requirements for training the STG model using our architecture.

6.1.1 Results of Hyperparameter Tuning

To achieve the best performance from our pipeline, we conducted the optimization procedure described in Section 5.5.3. The key metric of interest is $\mathcal{L}_{\text{DICs}}$, which needs to be minimized on the validation set, $\mathcal{T}_{\text{val}}$. Figure 6.1 reports the results for all 100 evaluated architectures. We observe that the architectural details, and the optimization ones, significantly influence performance. Note that the optimization procedure is stochastic (due to the optimizer), and ideally, this figure should include error bars to fully convey the variability. However, these error bars are inaccessible due to the high computational cost of training. A well-chosen architecture can improve the best loss value by nearly a factor of 10.

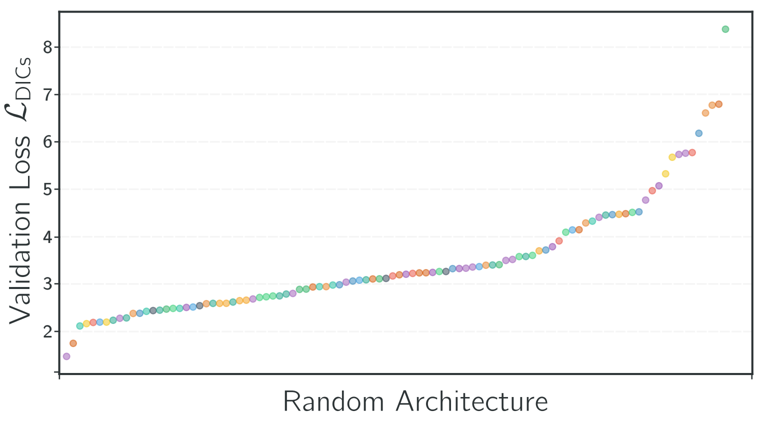


Figure 6.1: Performance comparison of random architectures. This scatter plot compares the best validation loss $\mathcal{L}_{\text{DICs}}$ for various random architectures, showcasing the variability in performance based on different architectural and optimization configurations. Each point represents a unique architecture, with the distribution of points illustrating how architectural choices affect model performance. The plot highlights the importance of selecting an appropriate architecture to achieve optimal performance, as evidenced by the spread of validation loss values across different configurations.

Next, we explore the commonalities among the best-performing architectures. Figure 6.2 shows both the *importance* of each hyperparameter and the *correlation* between their values and the loss (which we aim to minimize). [1]

The importance of a hyperparameter indicates how crucial its exact value is in determining whether the architecture performs well or poorly. Intuitively, it is essential to optimize important hyperparameters. Among the parameters measured as important, we find the learning rate, the encoder space dimension, the dropout value, and the batch size. Parameters of intermediate importance include the weights of the auxiliary tasks, the number of blocks in the encoder and decoder [2], and the latent space dimension. Finally, the number of heads in the multi-head attention layers and the activation function are measured as less important.

Among the notable correlations, the learning rate and dropout are the most pronounced. Both suggest that lower values are more suitable. For dropout, this is likely related to the amount of data and the various regularization mechanisms, which make it unnecessary. For the learning rate, this implies that convergence to good performance will probably be slower. Thus, for the final training, we increased the number of epochs to $N_{\text{epoch}} = 200$. From an architectural perspective, a shallow encoder with a high dimension is suitable, while the opposite is true for the decoder. For the auxiliary task weights, we observe that the classification task should be boosted, while the regression task should be diminished. This does not imply that regression is useless during training; this result can also be explained by the relative values of the losses. The regression loss is much bigger than the DICs loss or the classification loss, the latter being very small. We hypothesize that the identified values help balance these relative values over the long term of training.

Table 6.1a reports the final choice of hyperparameters for the architecture. This choice is based on the analysis above, combined with the importance of parameters to minimize the final model size. Thus, the

[1]Interested readers are redirected to Appendix F for details about how importance and correlation values are determined.

[2]This is a classic result for tasks where a small depth is sufficient, since residual connections are used, making identity mapping easy to learn. In other words, you can add blocks, but these are easily reduced to the point of not transforming the input.

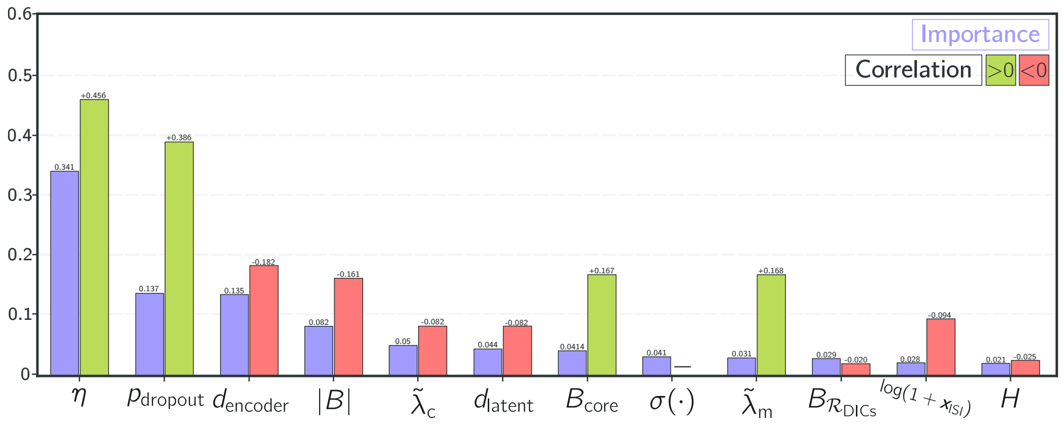


Figure 6.2: Importance and correlation of hyperparameters with model performance. This figure presents the importance of each hyperparameter (purple bars) and their correlation with the validation loss (green and red bars). The importance metric indicates how critical each hyperparameter is for determining the model performance, guiding the optimization process. The correlation metric shows how variations in hyperparameter values influence the loss, with green bars representing positive correlations and red bars indicating negative correlations. This analysis helps identify which hyperparameters most significantly impact model performance and how they should be tuned for optimal results.

Hyperparameter	**Final value**
Learning rate (η)	2.10×10^{-5}
Dropout (p_{dropout})	0.034
Latent space dimension (d_{latent})	16
Encoder space dimension (d_{encoder})	64
Number of heads (H)	8
Number of encoder blocks (B_{core})	4
Number of decoder blocks ($B_{\mathcal{R}_{\text{DICs}}}$)	2
Activation function ($\sigma(\cdot)$)	`gelu`
Apply log transform to input?	`true`
Regression weighting factor ($\tilde{\lambda}_m$)	0.0919
Classification weighting factor ($\tilde{\lambda}_c$)	5.44
Batch size ($\|B\|$)	32

(a) Final hyperparameter configuration for the optimized architecture. This table presents the final values of the hyperparameters selected after the tuning process. These values represent the configuration that achieved the best performance on the validation set according to the uncertainty-weighted DICs loss.

Number of Parameters	115,627

(b) Final model size. The final architecture, with 115,627 learnable parameters, is designed to be efficient and suitable for real-time applications, even on standard computational hardware.

Table 6.1: Summary of the optimized architecture for the STG neuron. This table summarizes the final hyperparameter configuration and the model size for the optimized architecture designed for the STG neuron. The configuration is optimized for performance and efficiency, making it suitable for real-time applications in experimental settings.

decoder dimension and depth are kept small, as the combination of the two is of low importance and allows for a smaller architecture.

The final architecture — before inserting adapters for the DA model, i.e., the model usable only for generating STG populations — has 115,627 parameters. It is worth noting that this is a small number for a DNN architecture, especially one with attention mechanisms. This is a positive aspect of our method, as it also means the architecture does not require high-performance hardware. Thus, it is feasible to run the model on any experimentalist's computer without needing a specific hardware (GPU, TPU, . . .), and in real-time with laboratory neuron recordings. Moreover, since this architecture is small and allows for fast inference, it can serve as an efficient building block for other architectures based on the theory of DICs in more complex tasks. We here think about the extension to neuronal networks.

To provide some benchmarks, as performance depends on the available hardware, training the pipeline takes less than 24 hours on an RTX 2080 Ti. For inference, the time for a forward pass per sample (averaged over 10,000 inferences of batches with 256 samples) is 1.25×10^{-5} seconds when the model is executed on a GPU and 6.24×10^{-5} seconds when executed on a CPU. These performance metrics indicates that the pipeline could be use directly in laboratory settings.

6.1.2 Results of Data Efficiency Analysis

Following the procedure described in Section 5.5.4, we evaluate the data requirements for training the model. This study is crucial because the bottleneck of the pipeline lies in its training phase, which is both computationally expensive and slow. Additionally, generating the data is time-consuming since CBMs are intractable, and solving the system of ODEs numerically is not fast, even with optimized and parallelized numerical procedures like those used in this work (Appendix B.3).

The architecture described previously (Table 6.1a) is evaluated after being trained on different fractions of the dataset, sampled to maintain good coverage of the DICs space. Figure 6.3 illustrates the obtained performances. To account for variability introduced by sampling, each dataset fraction is tested four times.

The general trend of the curve is classic: more data leads to better performance, with non-linear growth that eventually saturates. This means that beyond a certain training set size, the performance gain becomes marginal, and considering the training and generation cost, this gain can be deemed negligible. In our case, we observe that the acceptable fraction lies between $f = 0.25$ and $f = 0.5$. We choose the value $f = 0.4$ because it corresponds to a symbolic value where the average loss is less than 2. Thus, we will use a significantly smaller dataset for transferring to the DA neuron model.

This result is significant. The similar method by Gonçalves et al., 2020, based on DNNs, shows remarkable performance but requires a substantial amount of data for the full pipeline to be implemented. The authors report needing over 18 million samples for the STG model.[3] We argue that the theory of DICs is the primary reason for such a gain in data efficiency in our method. Indeed, we reduce the problem to point-wise regression in the space of DICs, compared with Gonçalves et al., which have to learn the entire joint distribution of conductances directly. In other words, Gonçalves et al. have to manage degeneracy in their predictions whereas our method handles degeneracy intrinsically.

Considering the fraction $f = 0.4$, our model requires 400,000 instances in the training set, which corresponds to 25,000 points in the DICs space. Although not explored in this work, it would be interesting to investigate the influence of the size of these 25,000 populations on performance. It is likely that populations of size less than $M = 16$ instances could be used to train the model, further reducing data requirements.

[3]For an honest comparison, it should be noted that Gonçalves et al., 2020 focuses on small STG circuits, rather than an isolated neuron. Also, a big part of their dataset is used only for a preprocessing step. Nevertheless, the entire dataset has been simulated.

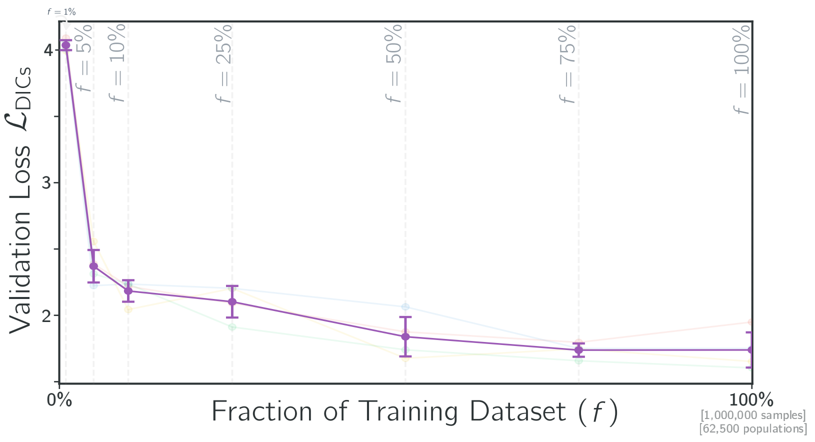


Figure 6.3: Test Loss as a function of training dataset size. This figure illustrates the relationship between the fraction of the training dataset (f) and the best test loss $\mathcal{L}_{\text{DICs}}$. The graph shows a decreasing trend in test loss as the fraction of the training dataset increases, indicating improved model performance with more data. Error bars represent the variability across multiple runs (4 per fraction), highlighting the consistency of the observed trend. The curve saturates beyond a certain dataset size, suggesting diminishing returns on performance gains with additional data.

6.2 Final Model Evaluation and Robustness Analysis

This section presents the final performance of the architecture trained with the best hyperparameters identified earlier. We evaluate the quality of predictions on the test data and analyze the model robustness against various levels of noise introduced into the input data.

It is important to note that the results are based on the test set, $\mathcal{T}_{\text{test}}$, and that this set has never been used in any choice related to the training or design.

6.2.1 Final Model Performance

We first report the performance in terms of the part of the loss function $\mathcal{L}_{\text{DICs}}$. We also evaluate the performance of the auxiliary tasks using balanced accuracy for classification and mean absolute error (MAE) for the regression of neuronal activity metrics. Table 6.2a reports these performances, which can be compared with the initial standard deviation present in the dataset, as reported in Table 6.2c.

One notable observation is that the encoder effectively constructs a meaningful latent representation of the input sequences. Both the classification and regression tasks are fulfilled efficiently by the model, which would not be possible without a robust latent representation from the encoder. Specifically, the classification accuracy is remarkably high at 99.83%. The regression tasks are also well-executed, with the number of spikes per burst — an integer value — predicted with an average error well below one. For other metrics, the average error is significantly lower than the dataset standard deviation, confirming successful learning and sufficient precision to accurately describe neuronal activity.

Furthermore, the loss value is close to the best values observed on the validation set, indicating good generalization capability of the model. Interpreting the loss value is complex because it cannot be directly translated into the ability to generate populations with activity close to the input activity. Additionally, the loss is weighted by uncertainty. Table 6.2b reports the MAE calculated on the predictions of DIC values, and Figure 6.4 illustrates these predictions. We observe a significant difference between the type of prediction for spiking inputs ($g_{\text{s}} > 0$) and bursting inputs ($g_{\text{s}} < 0$). Specifically, while bursting appears well-distinguished across the DICs space, spiking is predicted along a curve. Note that this does not reflect the quality of populations that could be generated. This is consistent with the idea of describing neuronal activity on different timescales: bursting requires three scales, whereas spiking requires only two.

Indeed, bursting activity inherently involves three distinct timescales, corresponding to fast spikes generation within bursts (g_{f}), intermediate modulation of bursting envelopes (g_{s}), and ultra-slow control of burst termination (g_{u}). This layered temporal structure allows bursting to be well-separated and distinctly mapped within the DICs space, as the model captures these multiple scales explicitly. In contrast, spiking activity is fundamentally governed by only two relevant timescales — a fast scale for spike generation (g_{s}) and a slower scale for recovery or adaptation (g_{s} and g_{u} that combine).

This simpler behavior for spiking arises because the model focuses exclusively on spike times, without incorporating the full voltage trace between spikes, and especially in the hyperpolarized region. Thus, both the slow and ultra-slow feedback are negative ones around the threshold voltage, and both play a similar role. One could probably talk about *degeneracy* in the DICs space for spiking instances. Indeed, because both are playing a negative feedback role, it is hard to extract solely from the spike times the exact contribution of each of them on the net negative feedback. As long as the net negative feedback arising from the slow and ultra-slow dynamic conductances has the right gain, corresponding to the recorded spiking frequency, their exact values don't matter; leading to more than one possible solution set, i.e. *degeneracy*.

Given Figure 6.4, the simple MAE metric on DIC values is not only difficult to interpret but likely meaningless unless bursting and spiking are separated. Additionally, using a loss function with uncertainty measures was particularly suitable for this situation, as the model assigns greater uncertainty to DIC values (particularly g_s) in spiking compared to bursting.

To obtain an interpretable metric, we follow the procedure described in Section 5.5.2. For each population in the test set, we select a representative instance, pass it through the pipeline, and use the predicted DICs to generate a new population. We then simulate this new population and calculate the distance ratio R_{VR}. This metric allows for an evaluation based on the generated populations — which is the ultimate goal of the pipeline — and is not directly dependent on the exact values of the predicted DICs. Figure 6.5 shows the results obtained in the DICs space. We observe that our pipeline is capable of achieving ratios below 1 in most of the DICs space (the lower, the better). This means that the population generated via the pipeline using an instance from a given degenerate population is, on average, closer in terms of neuronal activity to the picked instance than the original population is. This is strong evidence that our pipeline is effective in predicting DIC values associated with a given activity, thereby automatically generating populations with activity similar to the recorded one.

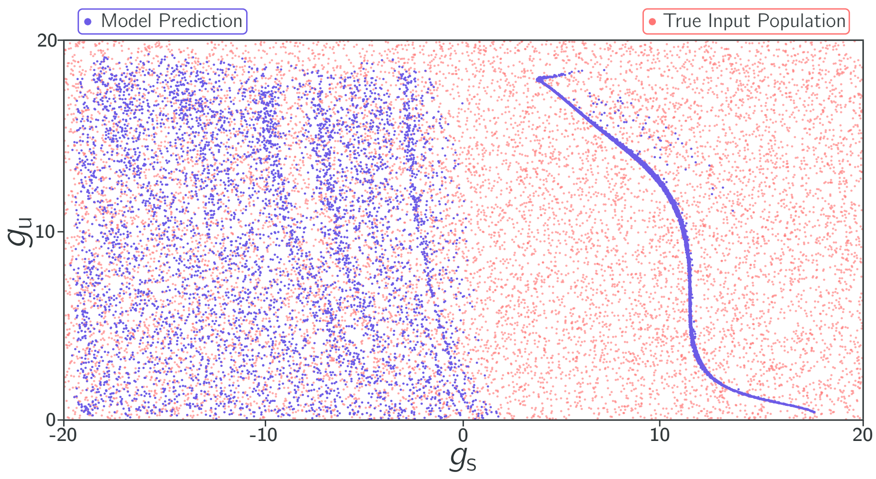
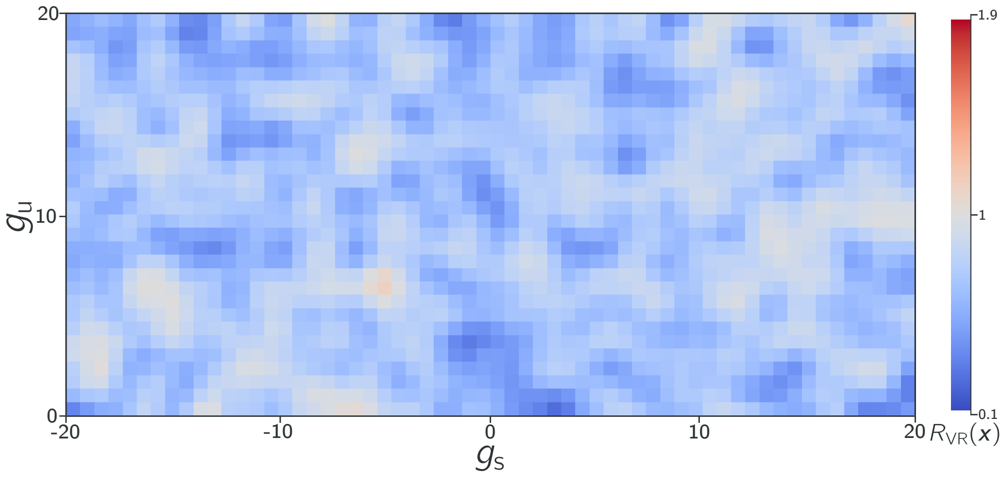
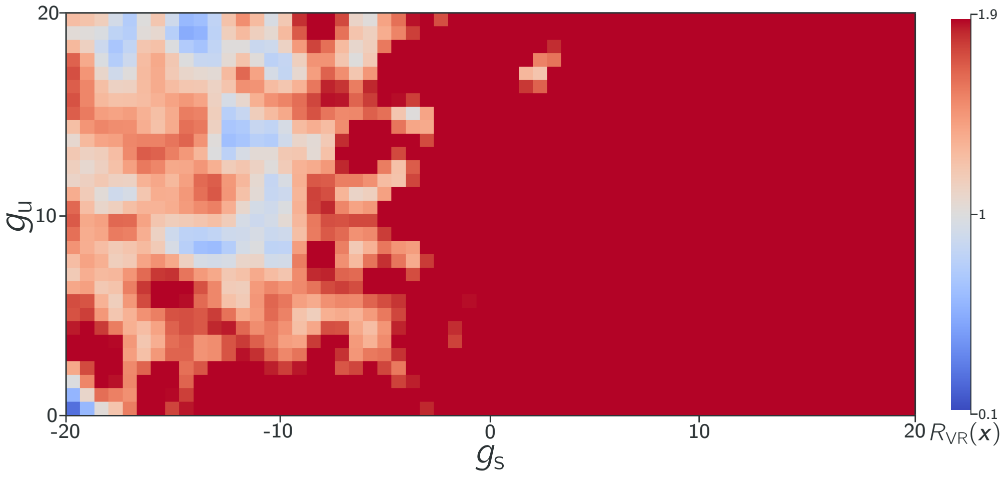


Figure 6.4: Distribution of model predictions in the DICs space of the STG model compared to the true distribution. This figure illustrates the distribution of model predictions (purple) in the Dynamic Input Conductances (DICs) space compared to the true distribution (red) of the test set, which was generated using Latin Hypercube Sampling (LHS). We observe a significant difference between the type of prediction for spiking inputs ($g_s > 0$) and bursting inputs ($g_s < 0$). Specifically, while bursting appears well-distinguished across the DICs space, spiking is predicted along a curve.

Finally, Figure 6.6 shows some examples of populations generated using the pipeline. We emphasize the degenerate nature of the population in Figure 6.7 where the distributions of the maximal conductances are provided. The developed tool effectively and automatically generates populations of CBMs with target activity based solely on the spike time sequence. Despite the low variability associated with the combination of prediction error from our pipeline and the intrinsic variability from the generation procedure (Section 2.4), the generated populations faithfully reproduce the activity of the input recording.



(A) Model-generated results using the predicted DICs.



(B) Results from a random guess in the DICs space.

Figure 6.5: Evaluation of the model performance in the DICs space of the STG model using the Van Rossum distance metric. This figure compares two evaluations: **(a)** the predicted DICs from the pipeline, and **(b)** a baseline using random guesses in the DICs space. Both use the Van Rossum distance metric on the generated populations to compute the performance, visualized as the distance ratio R_{VR}. The model-generated DICs produce significantly better results (lower R_{VR} values, closer to 0.1, in blue) compared to the random guess baseline. This highlights the model ability to predict meaningful DICs associated with realistic neuronal activity.

Metric	f_{spk} (MAE)	f_{intra} (MAE)	f_{inter} (MAE)	Duration($\boldsymbol{B}$) (MAE)	$\#\boldsymbol{B}$ (MAE)	Accuracy
Model Result	0.11 Hz	2.26 Hz	0.15 Hz	0.81 ms	0.16	99.83%

(a) Performance metrics of the backbone model on auxiliary tasks for the STG neuron. This table presents the performance of the backbone model on auxiliary tasks for the STG neuron, including the mean absolute error (MAE) for various metrics and the balanced classification accuracy. The metrics evaluated include spike frequency (f_{spk}), intra-burst frequency (f_{intra}), inter-burst frequency (f_{inter}), burst duration (Duration($\boldsymbol{B}$)), and the number of bursts ($\#\boldsymbol{B}$). The high accuracy and low MAE values indicate the model effectiveness in capturing the underlying patterns of neuronal activity for the STG neuron.

Metric	$\mathcal{L}_{\text{DICs}}$	g_{s} (MAE)	g_{u} (MAE)
Model Result	2.24	2.84	1.75

g_{s} (MAE - spiking)	g_{u} (MAE - spiking)	g_{s} (MAE - bursting)	g_{u} (MAE - bursting)
4.65	2.29	0.80	1.13

(b) Performance metrics of the backbone model on DIC predictions for the STG neuron. This table reports the performance of the backbone model in predicting Dynamic Input Conductances (DICs) for the STG neuron, including the overall loss $\mathcal{L}_{\text{DICs}}$ and the mean absolute error (MAE) for g_{s} and g_{u}. The table also breaks down the MAE for spiking and bursting scenarios, highlighting the model ability to distinguish between different types of neuronal activity. The results demonstrate the model capability to accurately predict DIC values for the STG neuron.

Metric (Standard Deviation)	f_{spk} (std)	f_{intra} (std)	f_{inter} (std)	Duration($\boldsymbol{B}$) (std)	$\#\boldsymbol{B}$ (std)
STG Dataset	3.02 Hz	47.91 Hz	1.11 Hz	7.85 ms	2.25

(c) Standard deviation of metrics in the dataset for the STG neuron. This table provides the standard deviation of various metrics in the dataset for the STG neuron, including spike frequency (f_{spk}), intra-burst frequency (f_{intra}), inter-burst frequency (f_{inter}), burst duration (Duration($\boldsymbol{B}$)), and the number of bursts ($\#\boldsymbol{B}$). These values serve as a baseline for evaluating the backbone model performance, allowing for a comparison between the natural variability in the data and the errors produced by the model for the STG neuron.

Table 6.2: Performance summary of the backbone model for the STG neuron. This table summarizes the backbone model performance on the test dataset for the STG neuron, including auxiliary task metrics, DIC predictions, and standard deviations of key metrics. The results demonstrate the model accuracy in predicting neuronal activity patterns and DIC values, highlighting its robustness and generalization capabilities.

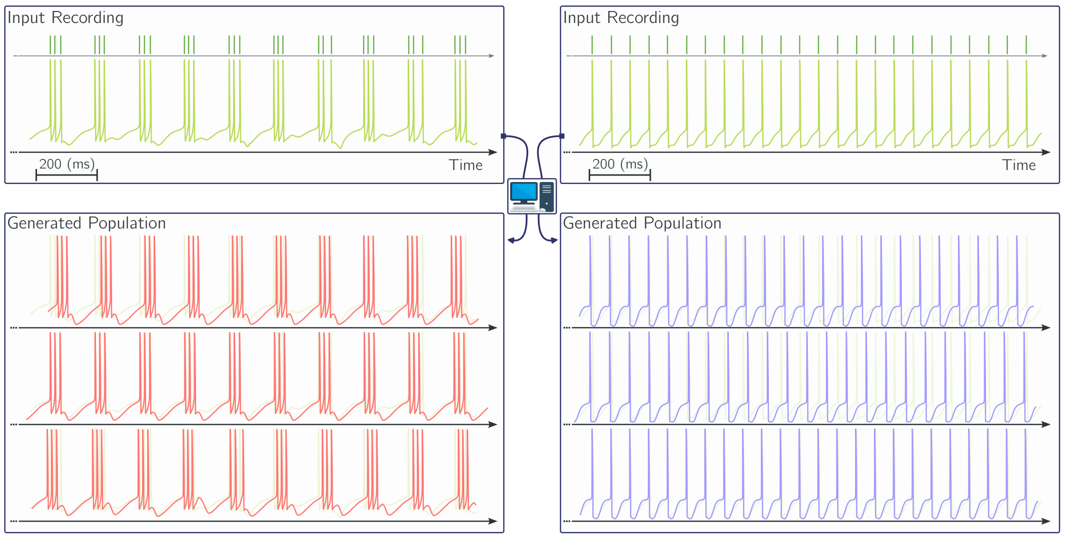


Figure 6.6: Comparison of input recordings and generated populations for the STG model. This figure illustrates the comparison between input neuronal recordings (top row) and the generated populations (bottom row) produced by the pipeline. The input recordings show the original spike time sequences, which are the only data provided to the model. The generated populations are created using the predicted DICs from the model, demonstrating the pipeline ability to reproduce neuronal activity patterns. The populations are degenerate and the maximal conductance distributions can be seen in Fig. 6.7.

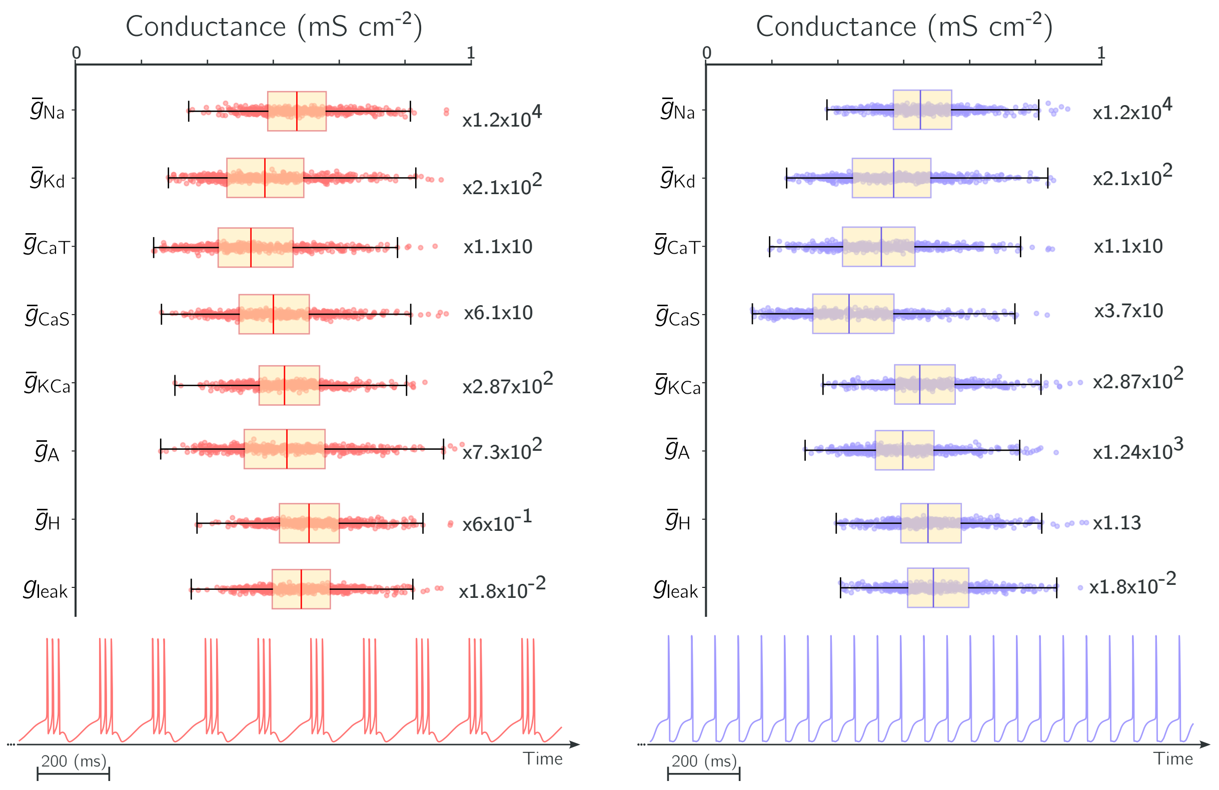


Figure 6.7: Comparison of conductance distributions in the generated spiking and bursting degenerate populations for the STG model. This figure presents the distributions of maximal conductances. The box plots illustrate the variability and central tendency of conductance values, with the numerical multiplier indicating the scaling factor that should be applied to get the true values. The bottom traces show representative for each population (500 instances). The strong variability highlights the degenerate nature of the generated populations.

6.2.2 *Impact of Input Noise*

Noise is an intrinsic component when dealing with neuronal activities and their recordings. Robust tools are essential for handling this noise. Several sources of noise with varying intensities can be identified, including the inherent stochasticity of neuronal firing and noise related to the environment and sensors (Mainen and Sejnowski, 1995; Yang et al., 2009).

We propose studying the robustness of our pipeline by observing error variations on the test set when noise is injected into the spike time sequences. Note that this analysis is basic, and the best method would involve testing the pipeline on data actually recorded in a laboratory setting. An intermediate step could involve using a more biologically realistic noise model rather than simple Gaussian noise (Goldwyn and Shea-Brown, 2011).

Figure 6.8 illustrates the results for different noise intensities. We observe minimal variation in performance, indicating the robustness of our method within the studied noise ranges. In practice, we even observe a reduction in loss for intermediate noise values. This result may seem counterintuitive but actually highlights the model ability to quantify its uncertainty (since the loss is weighted by predicted uncertainty). The variation in the MAE of g_s and g_u allows us to assess the impact of the model uncertainty, and we observe that the model predictions on spiking are not affected by noise. For bursting, noise has a slight impact. This result is consistent with the timescales of bursting activity. Specifically, noise with an intensity of $\sigma_{\text{noise}} = 5\,\text{ms}$ is on the order of magnitude of part of the bursting dataset, particularly for the intra-burst period (Fig. 5.6).

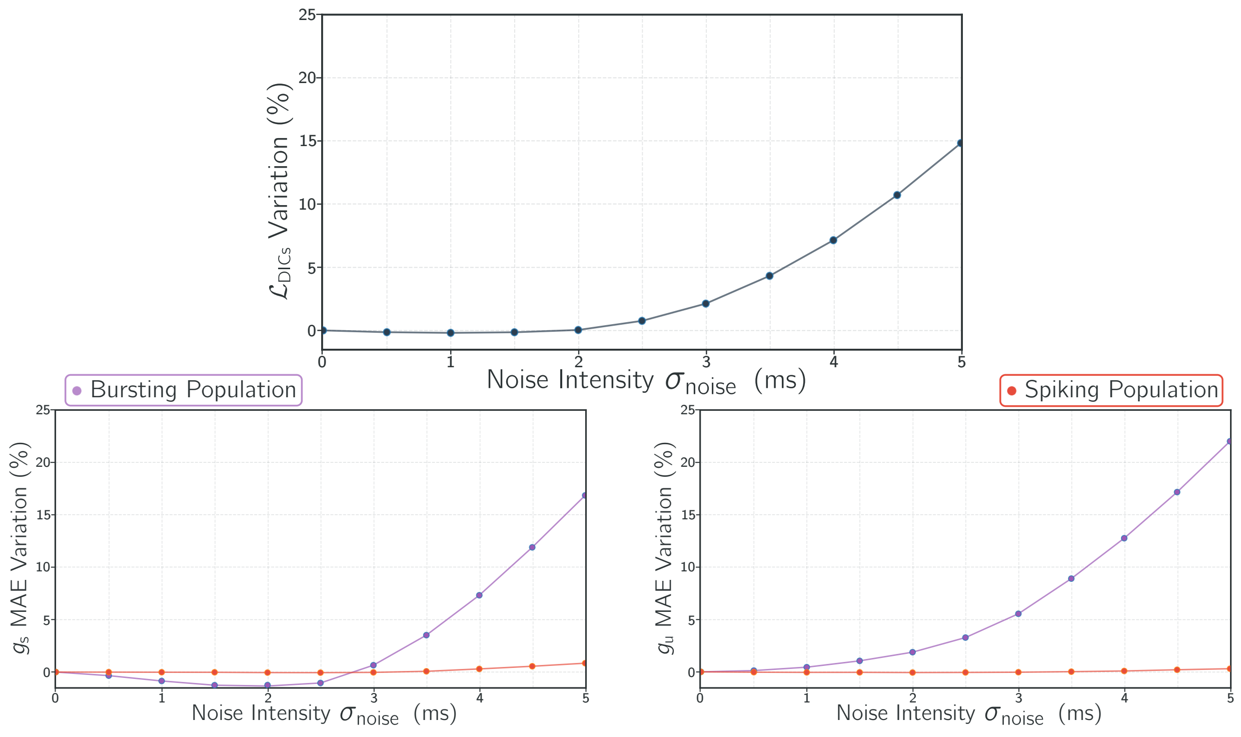


Figure 6.8: Impact of noise intensity on model performance for spiking and bursting populations. This figure illustrates the effect of varying noise intensities (σ_{noise}) on the performance of the model, measured by the variation in $\mathcal{L}_{\text{DICs}}$ and the mean absolute error (MAE) for g_s and g_u. The top plot shows the overall variation in $\mathcal{L}_{\text{DICs}}$ across the entire test set, relative to the performance without noise. The bottom plots provide a detailed view of the MAE for g_s (left) and g_u (right) for bursting (purple) and spiking (red) populations across different noise intensities. The results indicate that the model is robust to noise, with minimal performance degradation for spiking populations and a slight impact on bursting populations, particularly at higher noise levels. This robustness is crucial for the model applicability in experimental settings where noise is inherent.

To further support the validity of our results, we also conduct a qualitative evaluation by presenting the pipeline with a Poisson process — a model frequently used in neuroscience as a proxy for real experimental data (Dayan and Abbott, 2001). A Poisson process is characterized by much greater variability in spike timing than what the STG and DA models typically generate. Figure 6.9A illustrates one of the populations generated from a representative Poisson spiking process. We observe that the pipeline remains stable and produces qualitatively satisfactory results despite the high variability in the input. The average firing rate appears to be captured effectively by the pipeline. Figure 6.9B shows the results for a representative Poisson bursting process, which are more than satisfactory.

In summary, our pipeline shows promise in an experimental context due to its robustness to noise.

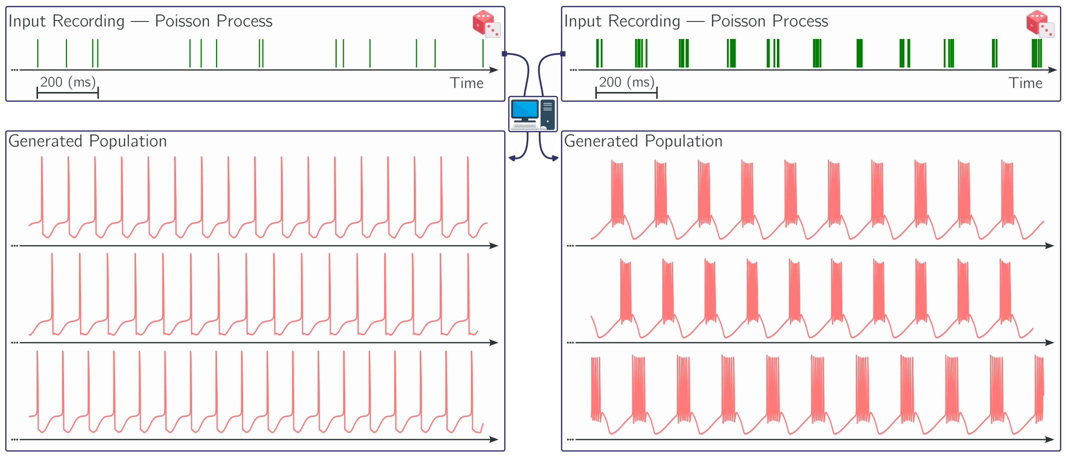


Figure 6.9: Comparison of generated populations from Poisson spiking and bursting processes. This figure illustrates the results of the pipeline when presented with Poisson processes, which are used as proxies for real experimental data. The left panel (A) shows a population generated from a representative Poisson spiking process, demonstrating the pipeline ability to capture the average firing rate despite high variability in spike timing. The right panel (B) displays a population generated from a representative Poisson bursting process, highlighting the pipeline effectiveness in producing qualitatively satisfactory results even with increased input variability. These results underscore the robustness and potential applicability of the pipeline in experimental contexts.

6.3 Results of Transfer to the Dopaminergic Neuron Model

In this section, we present the experimental results obtained when adapting the architecture to the dopaminergic neuronal model using LoRA adapters.

6.3.1 Optimization of LoRA Parameters

LoRA adapters can be characterized by a single hyperparameter, r, which encodes the intermediate dimension and makes these adapters "low-rank." We study the influence of r following a grid-search procedure (Appendix G). Table 6.3[4] reports the best validation loss associated with each value of r, along with the number of *new* parameters introduced by the adapters. Note that the number of parameters follows a linear relationship with r:

$$\#\theta_{\text{LoRA}} = \underbrace{1245r}_{\text{LoRA } A \text{ and } B \text{ matrices}} + \underbrace{4}_{\boldsymbol{\mu}_{\text{train}} \text{ and } \boldsymbol{\sigma}_{\text{train}}} \tag{6.1}$$

r	2	4	8	16	**32**	48	64
$\mathcal{L}_{\text{DICs}}$	3.926	2.964	2.596	1.979	1.633	1.935	1.721
$\#\theta_{\text{LoRA}}$	2494	4984	9964	19,924	39,844	59,764	79,684

Table 6.3: Performance and parameter count for different values of r used for the LoRA. This table presents the relationship between the rank r, the loss $\mathcal{L}_{\text{DICs}}$, and the number of LoRA parameters $\#\theta_{\text{LoRA}}$. The data illustrates how varying the rank r affects both the model performance and the number of parameters required for the LoRA adaptation.

We use the value $r = 32$, which yields the best results. Interestingly, this dimension lies between d_{encoder} and d_{latent}. By following this method, we are able to extend our pipeline to a new CBM by adding only $39,844$ parameters, which is approximately 34% of the number of parameters we would have needed to store and manipulate if training the pipeline from scratch. Training is also less costly since fewer parameters need to be updated. The final adapters of the DA model are trained for $N_{\text{epoch}} = 200$ epochs.

6.3.2 Evaluation of Transfer Performance

This section aims to demonstrate that using adapters is an effective method with a favorable trade-off between the number of parameters and performance. To do this, we compare the performance of the adapters with that of a model trained from scratch. Finally, we will illustrate the performance in the DICs space of the DA model and show some traces of generated populations.

Table 6.4 reports the complete results and standard deviations of various metrics in the DA model dataset for comparison purposes. Tables 6.4a and 6.4b are related to the architecture trained from scratch (i.e., without adapters, meaning an architecture that has completely forgotten how to predict on the STG model). In contrast, Tables 6.4c and 6.4d pertain to the architecture with the LoRA adapters (i.e., an architecture that is still capable of handling the STG model if the adapters are deactivated).

The first observation is that our method can be easily generalized to new CBMs. The performances are not due to chance on the STG model, since we can also obtain them on the DA model. We argue that our method is very general and could be applied without issue to any type of model that is interpretable and generable through the theory of DICs. The only constraint is the availability of data, but since the data are synthetic, they can be easily generated.

[4] Note that the loss values, $\mathcal{L}_{\text{DICs}}$, should not be compared across different CBMs, but rather among models focusing on the same CBM.

Next, we observe that although the performance of the model trained from scratch is slightly better, the adapted model presents very close results. We conclude that this technique is effective for extending the pipeline at a lower cost. The major difference seems to be in the results of the auxiliary regression task, and we hypothesize that fine-tuning the encoder could potentially yield equivalent or even better performance with the adapters. Our primary interest is in the main task of regressing DIC values, and this task is effectively accomplished by the model with its adapters.

As with the STG model, we observe differences in the MAE for DIC values between spiking and bursting instances. This confirms that the separation of timescales (two for spiking and three for bursting) is a general phenomenon and not specific to the STG model. We emphasize again that the MAE value in the DICs space does not directly interpret the quality of the generated populations. A small MAE value is indicative of a well-functioning pipeline, but a large MAE value is more complex to interpret, especially in the case of spiking.

Figure 6.10 represents the distribution of the model outputs in the DICs space. Once again, we observe a difference in behavior between the predictions for spiking and bursting. Interestingly, the left-hand branch extending downwards corresponds to instances classified as spikers but with a very high average frequency — the fast spikers. Figure 6.11 shows the map of the van Rossum distance ratios, which is more interpretable than the MAE on the predictions. Figure 6.12 illustrates populations generated using our method for the DA model and Figure 6.13 shows the conductances distributions, highlighting the degenerate nature of the generated populations. We observe that the pipeline is capable of generating populations with activity similar to that of the input. Consequently, we claim that our method is easily generalizable to new CBMs.

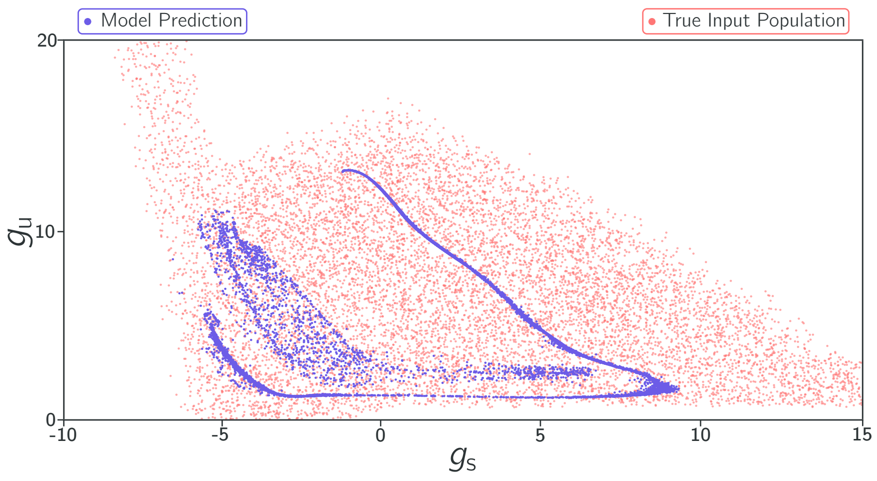


Figure 6.10: Distribution of model predictions in the DICs space of the DA model compared to the true distribution. This figure illustrates the distribution of model predictions (purple) in the Dynamic Input Conductances (DICs) space compared to the true distribution (red) of the test set, which was generated using Latin Hypercube Sampling (LHS). As for the STG model (Fig. 6.4), bursting appears well-distinguished across the DICs space, spiking is predicted along a curve. *Silent populations have been removed.*

Metric	f_{spk} (MAE)	f_{intra} (MAE)	f_{inter} (MAE)	Duration($\boldsymbol{B}$) (MAE)	$\#\boldsymbol{B}$ (MAE)	Accuracy
Model Result	1.57 Hz	2.43 Hz	0.60 Hz	39.64 ms	2.65	99.55%

(a) Performance metrics of the architecture on auxiliary tasks for the DA neuron trained from scratch. This table presents the performance of the architecture on auxiliary tasks for the DA neuron, including the mean absolute error (MAE) for various metrics and the balanced classification accuracy. The metrics evaluated include spike frequency (f_{spk}), intra-burst frequency (f_{intra}), inter-burst frequency (f_{inter}), burst duration (Duration($\boldsymbol{B}$)), and the number of bursts ($\#\boldsymbol{B}$).

Metric	$\mathcal{L}_{\text{DICs}}$	g_{s} (MAE)	g_{u} (MAE)
Model Result	1.16	2.36	1.01

g_{s} (MAE - spiking)	g_{u} (MAE - spiking)	g_{s} (MAE - bursting)	g_{u} (MAE - bursting)
2.57	1.08	1.25	0.61

(b) Performance metrics of the architecture on DIC predictions for the DA neuron trained from scratch. This table reports the performance of the architecture in predicting Dynamic Input Conductances (DICs) for the DA neuron, including the overall loss $\mathcal{L}_{\text{DICs}}$ and the mean absolute error (MAE) for g_{s} and g_{u}. The table also breaks down the MAE for spiking and bursting scenarios, highlighting the model ability to distinguish between different types of neuronal activity.

Metric	f_{spk} (MAE)	f_{intra} (MAE)	f_{inter} (MAE)	Duration($\boldsymbol{B}$) (MAE)	$\#\boldsymbol{B}$ (MAE)	Accuracy
Model Result	1.67 Hz	2.94 Hz	0.44 Hz	30.13 ms	2.63	99.75%

(c) Performance metrics of the architecture with LoRA adapters on auxiliary tasks for the DA neuron. This table presents the performance of the architecture with LoRA adapters on auxiliary tasks for the DA neuron, including the mean absolute error (MAE) for various metrics and the balanced classification accuracy. The metrics evaluated include spike frequency (f_{spk}), intra-burst frequency (f_{intra}), inter-burst frequency (f_{inter}), burst duration (Duration($\boldsymbol{B}$)), and the number of bursts ($\#\boldsymbol{B}$).

Metric	$\mathcal{L}_{\text{DICs}}$	g_{s} (MAE)	g_{u} (MAE)
Model Result	1.37	2.41	1.10

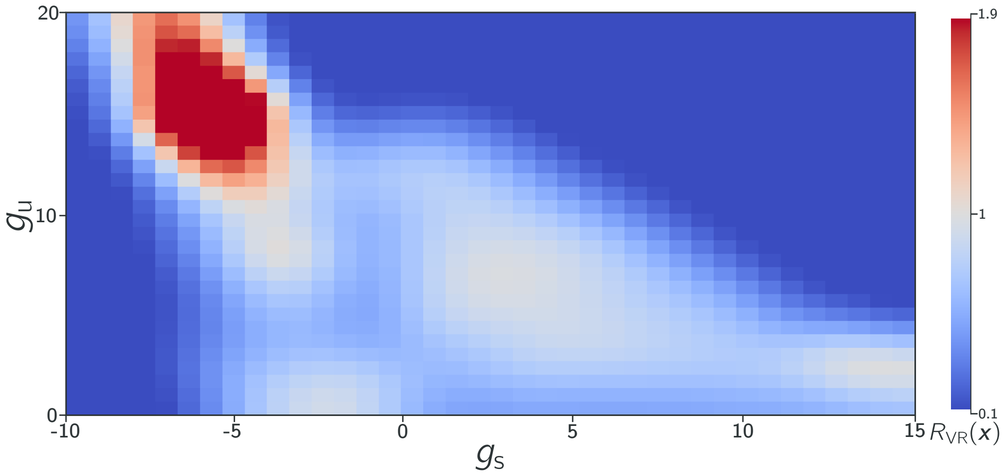
g_{s} (MAE - spiking)	g_{u} (MAE - spiking)	g_{s} (MAE - bursting)	g_{u} (MAE - bursting)
2.62	1.18	1.28	0.68

(d) Performance metrics of the architecture with LoRA adapters on DIC predictions for the DA neuron. This table reports the performance of the architecture with LoRA adapters in predicting Dynamic Input Conductances (DICs) for the DA neuron, including the overall loss $\mathcal{L}_{\text{DICs}}$ and the mean absolute error (MAE) for g_{s} and g_{u}. The table also breaks down the MAE for spiking and bursting scenarios, demonstrating the model capability to accurately predict DIC values.

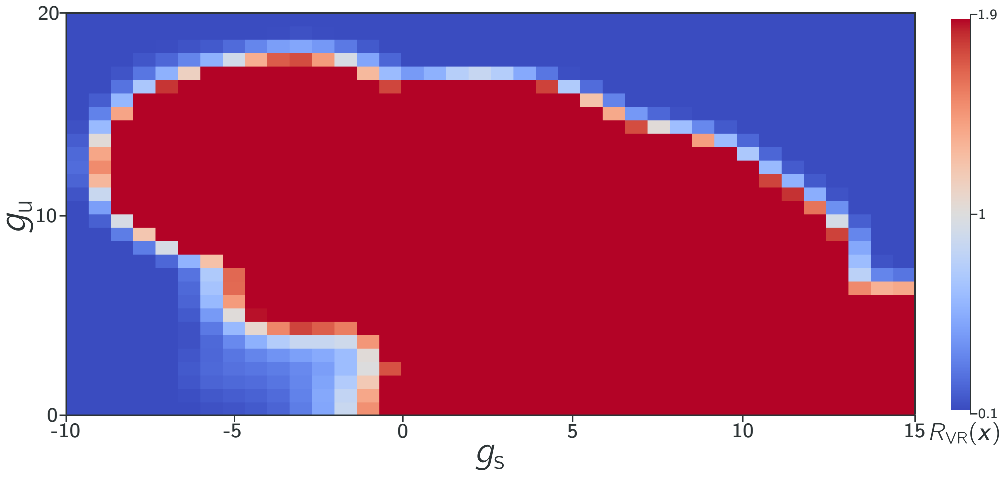
Metric (Standard Deviation)	f_{spk} (std)	f_{intra} (std)	f_{inter} (std)	Duration($\boldsymbol{B}$) (std)	$\#\boldsymbol{B}$ (std)
DA Dataset	19.85 Hz	14.5 Hz	1.18 Hz	296 ms	15.64

(e) Standard deviation of metrics in the dataset for the DA neuron. This table provides the standard deviation of various metrics in the dataset for the DA neuron, including spike frequency (f_{spk}), intra-burst frequency (f_{intra}), inter-burst frequency (f_{inter}), burst duration (Duration($\boldsymbol{B}$)), and the number of bursts ($\#\boldsymbol{B}$). These values serve as a baseline for evaluating the architecture performance.

Table 6.4: Performance summary of the architecture for the DA neuron. This table summarizes the architecture performance on the test dataset for the DA neuron, including auxiliary task metrics, DIC predictions, and standard deviations of key metrics. The results demonstrate the effectiveness of LoRA adapters in achieving performance close to training from scratch, highlighting the robustness and generalization capabilities of the approach.



(A) Model-generated results using the predicted DICs. The bad result area at top left corresponds to a region with very few instances in the dataset. This is an area of high entropy, with many silent and fast-spiking instances, considered pathological. *The typical activities of the DA neuron are not located in this region.*



(B) Results from a random guess in the DICs space.

Figure 6.11: Evaluation of the model performance in the DICs space of the DA model using the Van Rossum distance metric. This figure compares two evaluations: **(a)** the predicted DICs from the pipeline, and **(b)** a baseline using random guesses in the DICs space. Both use the Van Rossum distance metric to compute the performance, visualized as the distance ratio R_{VR}. The model-generated DICs produce significantly better results (lower R_{VR} values, close to 1 are considered as good results since this correspond to the baseline variability of the generation procedure) compared to the random guess baseline. Our method can be generalized to different CBMs.

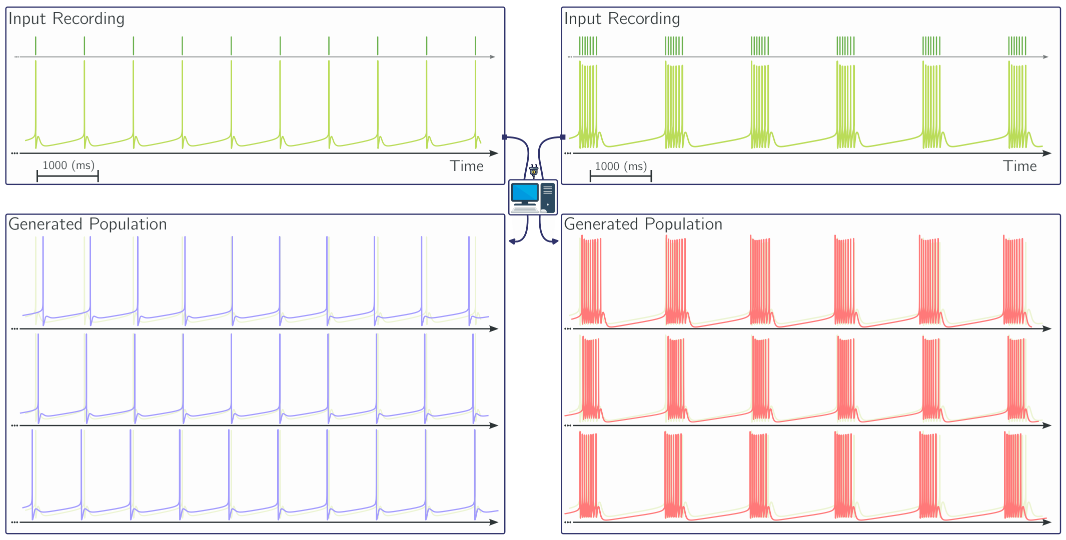


Figure 6.12: Comparison of input recordings and generated populations for the DA model. This figure illustrates the comparison between input neuronal recordings (top row) and the generated populations (bottom row) produced by the pipeline. The input recordings show the original spike time sequences, which are the only data provided to the model. The generated populations are created using the predicted DICs from the model, demonstrating the pipeline ability to reproduce neuronal activity patterns.

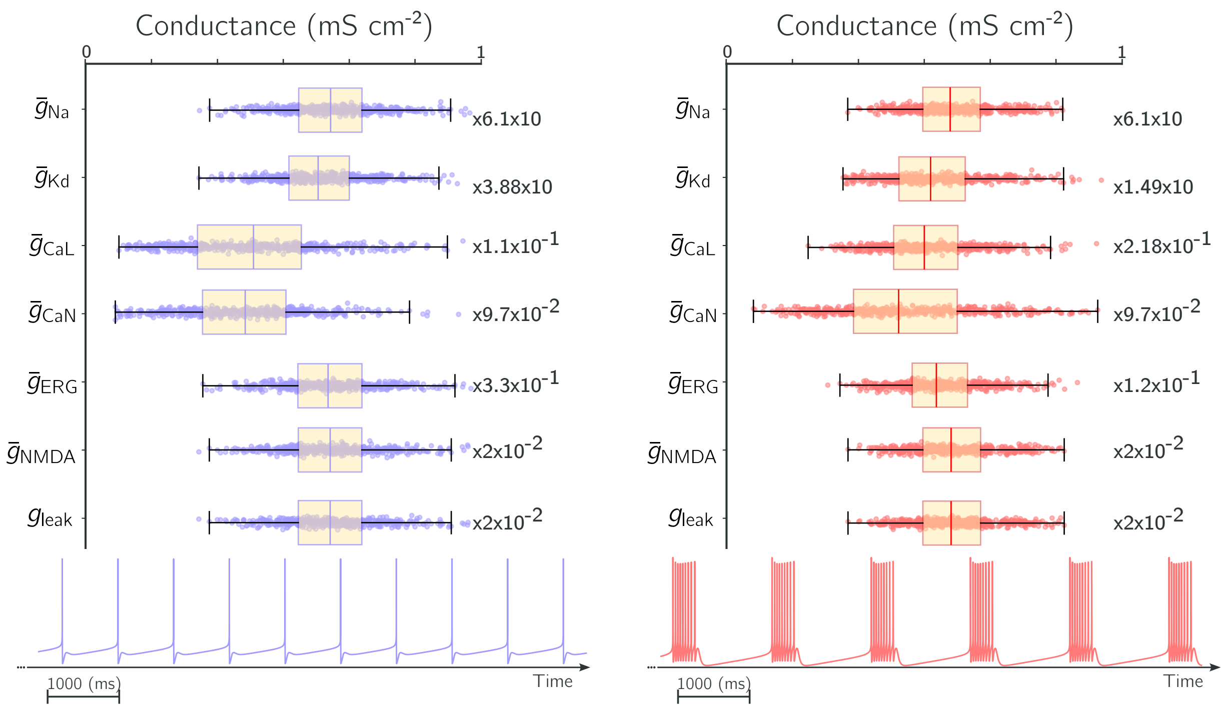


Figure 6.13: Comparison of conductance distributions in the generated spiking and bursting degenerate populations for the DA model. This figure presents the distributions of maximal conductances. The box plots illustrate the variability and central tendency of conductance values, with the numerical multiplier indicating the scaling factor that should be applied to get the true values. The bottom traces show representative for each population (500 instances). The strong variability highlights the degenerate nature of the generated populations.

Chapter 7

Development of an Open-Source Software for Computational Neuroscience

This chapter briefly presents the application developed based on the findings of this thesis. The primary objective behind developing this software is to *truly* and *practically* address the central question of this work: "*How can we reconcile experimentalists with computational neuroscience?*"

The idea is that experimentalists come from diverse backgrounds, and coding is not a universal skill. Therefore, we propose a simple application that can be used directly on a computer in the laboratory, serving as an interface to the developed pipeline. The application automates the inference process through the pipeline, facilitating the generation of populations from DIC values, which are internally managed.[1]

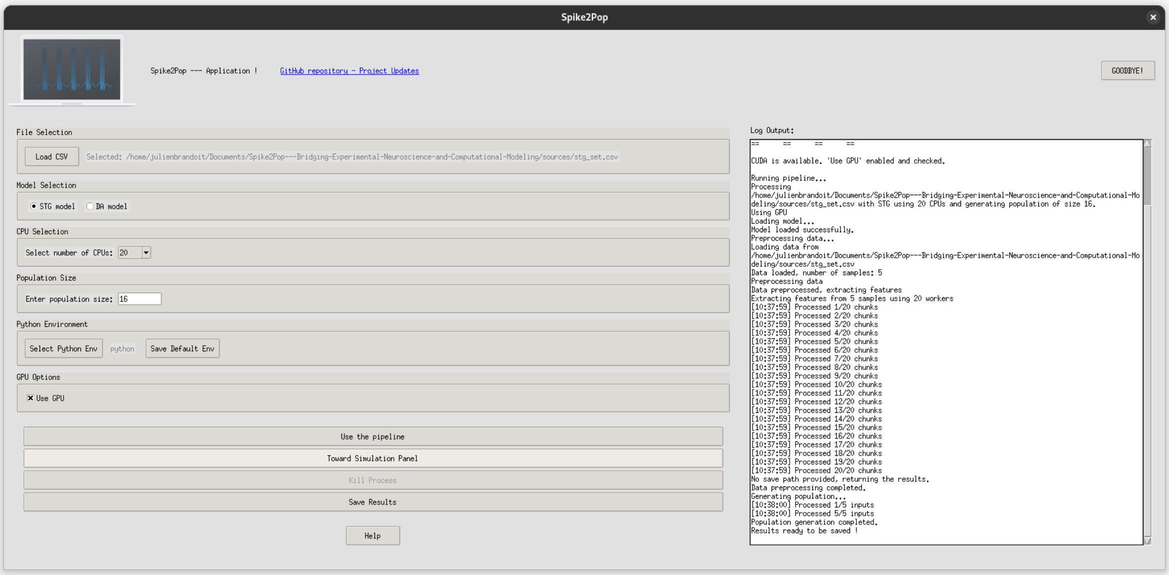
Figure 7.1 shows the application interface, which is widely accessible without technical expertise. For example, in just a few seconds on a personal computer, it is capable of generating the conductances of over 5,000 populations of 16 instances from different experiments.

Thus, the user can provide a CSV file with all the spike time sequences recorded in the laboratory, each associated with an ID. With just a few clicks on a graphical interface, they can obtain and save a CSV file on their computer containing all the generated populations — of the desired size — associated with the original ID. The conductance values can be used in the simulator of their choice or through the codes developed for this thesis. The application efficiently utilizes available hardware with automatic detection of multiple CPUs and a GPU, if available.

Additionally, the application includes a simulation panel, as shown in Figure 7.2A, which allows experimentalists to simulate the generated populations directly within the software. This feature enables users to visualize the results, including the traces, the original recordings of spike times, and the conductance distributions, thereby highlighting the degeneracy as illustrated in Figure 7.2B. Users can also export the simulation traces for further analysis or reporting.

This application serves as a proof of concept that it is possible to develop not only new methods but also tools that make these methods available, accessible, and practical for experimentalists. It could also serve as an easy entry point for validating the method on laboratory data.

[1]The application repository is available at:
https://github.com/julienbrandoit/Spike2Pop---Bridging-Experimental-Neuroscience-and-Computational-Modeling.



Spike2Pop

Spike2Pop --- Application ! GitHub repository - Project Updates

File Selection

Selected: /home/julienbrandoit/Documents/Spike2Pop---Bridging-Experimental-Neuroscience-and-Computational-Modeling/sources/stg_set.csv

Model Selection

[✓ STG model] DA model

CPU Selection

Select number of CPUs: 20

Population Size

Enter population size: 16

Python Environment

python

GPU Options

[✓ Use GPU]

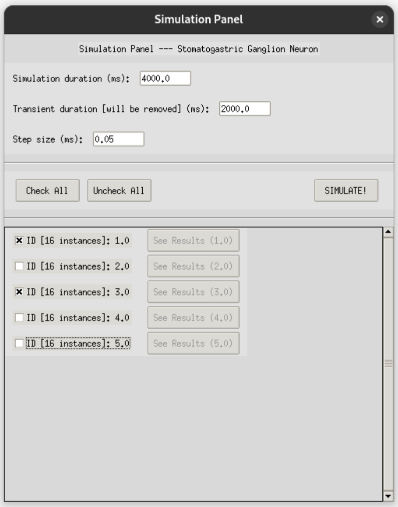
Log Output:

```
==      ==      ==      ==

CUDA is available. 'Use GPU' enabled and checked.

Running pipeline...
Processing
/home/julienbrandoit/Documents/Spike2Pop---Bridging-Experimental-Neuroscience-and-Computational-Mo
deling/sources/stg_set.csv with STG using 20 CPUs and generating population of size 16.
Using GPU
Loading model...
Model loaded successfully.
Preprocessing data...
Loading data from
/home/julienbrandoit/Documents/Spike2Pop---Bridging-Experimental-Neuroscience-and-Computational-Mo
deling/sources/stg_set.csv
Data loaded, number of samples: 5
Preprocessing data
Data preprocessed, extracting features
Extracting features from 5 samples using 20 workers
[10:37:59] Processed 1/20 chunks
[10:37:59] Processed 2/20 chunks
[10:37:59] Processed 3/20 chunks
[10:37:59] Processed 4/20 chunks
[10:37:59] Processed 5/20 chunks
[10:37:59] Processed 6/20 chunks
[10:37:59] Processed 7/20 chunks
[10:37:59] Processed 8/20 chunks
[10:37:59] Processed 9/20 chunks
[10:37:59] Processed 10/20 chunks
[10:37:59] Processed 11/20 chunks
[10:37:59] Processed 12/20 chunks
[10:37:59] Processed 13/20 chunks
[10:37:59] Processed 14/20 chunks
[10:37:59] Processed 15/20 chunks
[10:37:59] Processed 16/20 chunks
[10:37:59] Processed 17/20 chunks
[10:37:59] Processed 18/20 chunks
[10:37:59] Processed 19/20 chunks
[10:37:59] Processed 20/20 chunks
No save path provided, returning the results.
Data preprocessing completed.
Generating population...
[10:38:00] Processed 1/5 inputs
[10:38:00] Processed 5/5 inputs
Population generation completed.
Results ready to be saved !
```

Figure 7.1: Application Interface. The screenshot of the user-friendly interface of the application, designed for easy access and use by experimentalists without requiring technical expertise.



Simulation Panel

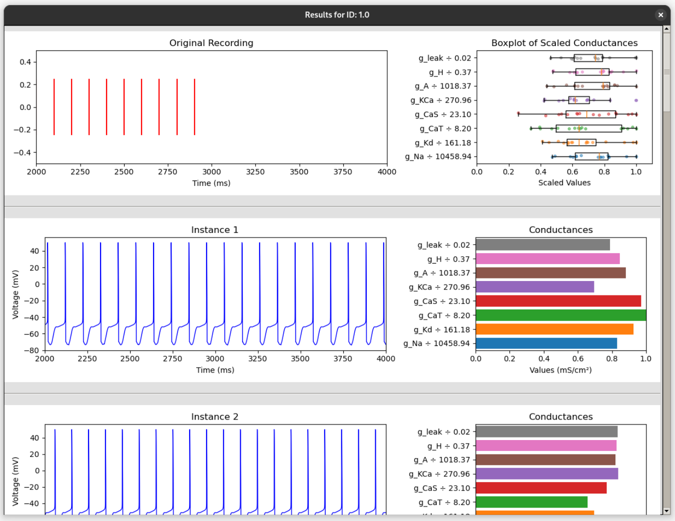
Simulation Panel --- Stomatogastric Ganglion Neuron

Simulation duration (ms): 4000.0

Transient duration [will be removed] (ms): 2000.0

Step size (ms): 0.05

[✓ ID [16 instances]: 1.0]
ID [16 instances]: 2.0
[✓ ID [16 instances]: 3.0]
ID [16 instances]: 4.0
ID [16 instances]: 5.0



(A) Simulation Panel. The interface panel where users can simulate populations.

(B) Results Panel. An example of the visualization output showing the traces, original spike time recordings, and conductance distributions, highlighting the degeneracy in the generated populations.

Figure 7.2: Simulation and Results Panels in the Spike2Pop Application.

Chapter 8

Limitations and Perspectives

This chapter serves as a conclusion to this work. It explores and highlights the limitations of the designed method as well as the perspectives for broader applications or key experiments that could be conducted.

8.1 Biological Applications and Insights

A first observation is that the proposed pipeline is both fast and robust to noise, while also being user-friendly thanks to the developed software. Its performance on *in silico* experiments has been highly satisfactory. This opens promising perspectives for applying the pipeline to real experimental data in laboratory settings. Such use would serve two goals: validating the pipeline applicability under realistic conditions and confirming its alignment with the needs and expectations of experimentalists.

The pipeline requirements are minimal — it only needs spike time data, which is commonly recorded in a wide range of neuronal experiments. As such, its application spans numerous experimental contexts.

For instance, one could investigate how various neuromodulators or external stimuli affect ion channel composition by comparing spike time sequences recorded before and after such perturbations. Analyses can be conducted in the DICs space or directly on the inferred maximal conductances. These experiments can be complemented by interpretation through the reachability maps developed in this work. One can imagine identifying more specifically which conductances are not involved in the observed changes.

Although this thesis primarily focuses on identifying degenerate sets of maximal conductances associated with a fixed behavior — effectively a static set — the framework naturally extends to non-stationary behaviors. Some transient activities can be broken down into a succession of spike time sequences assumed to be static (as a quasi-static hypothesis). In such cases, it becomes possible to trace a trajectory through parameter space, reflecting how conductances evolve alongside neural activity.

Despite its strengths, the pipeline is inherently limited by design choices. Notably, it simplifies neuronal dynamics by representing them solely through spike times. While this enables broad analysis (e.g., Heiligenberg, 1991; Johnson and Hsiao, 1992; Seidemann et al., 1996; Sharma et al., 2022), it excludes sub-threshold dynamics and other features of neuronal activity (e.g., Engel et al., 2008; Lampl and Yarom, 1993; Valero et al., 2022).

Future improvements could involve predicting additional points along the DIC curves, such as hyperpolarized zones or depolarized states (e.g., *up-state potentials*, as studied in Drion et al., 2015). The pipeline architecture is well-suited for integrating such features, whether temporal or global, through its encoder. Techniques like conditional neural processes (Garnelo et al., 2018) may offer a path toward fully probabilistic and complete predictions of DIC curves. A more detailed description of these curves would enhance control over the trade-off between the *level of degeneracy* and the *degenerate proximity* of the solutions. Moreover, incorporating a stimulation current $I_{\text{ext}}(t)$ into the pipeline input would be a valuable extension.

Expanding the method to neural networks, beyond single-cell models, is another compelling avenue for future research in computational neuroscience.

Additional limitations include the assumption that the CBM formulation is exact. In its current form, the method cannot accommodate uncertainty in the governing equations — such as ion channel kinetics — which may not be precisely known. Furthermore, the generation process requires specifying *a priori* distributions for conductances. Yet, the literature lacks comprehensive studies on biologically plausible parameter ranges. Better empirical knowledge would allow more accurate reachability maps, perhaps by imposing biologically-informed upper bounds on conductance values after compensation.

While the final architecture is lightweight, efficient, and can run on a personal computer, generalizing it to new CBMs necessitates training new LoRA-based adapters. This process depends on simulating synthetic data, as generalization cannot be achieved directly from CBM equations. However, once trained, these adapters allow for straightforward application. Compared to methods such as Gonçalves et al., 2020, our approach is notably more data-efficient.

8.2 Neuromorphic Systems: Extending Beyond Biology

In addition to the biological applications discussed, the methodologies and insights gained from this work could have significant implications for neuromorphic systems. These systems, inspired by biological neural networks, aim to mimic neuro-biological architectures to enhance computational efficiency and adaptability (Indiveri et al., 2011; Kudithipudi et al., 2025). Although promising, these systems are constrained by a phenomenon affecting all analog integrated circuits, which introduces *mismatches* in the transistors. That is, despite identical properties on paper, the error margins, due to manufacturing tolerances and the variations in the silicon substrate, cause the behavior of the fabricated systems to vary, sometimes significantly.

Recent developments by Mendolia et al., 2025 propose a new silicon neuron — that is, the hardware implementation of a neuron model, built on an electronic chip — designed and modelled through the DICs theory. Thus, even if the team of Mendolia et al. imposes identical voltages — equivalent to feedback gains, i.e., the conductances of biological neurons — associated with precise DIC values, and thus with a specific activity; the neurons all behave differently.

As for conductance-based models in the early days of computational neuroscience, the use of neuromorphic hardware requires long and tedious procedures of hand-tuning, limiting their potential impact.

Since these new models are based on the DICs theory, we argue that the method developed in this work could be used to correct *a posteriori* the problems related to mismatches. For example, we suggest that a recalibration procedure could extract bias values between the imposed DIC values and the truly recorded values. The implementation of a controller on the chip (for example, via a neuromorphic translation of the work of Fyon et al., 2023) could correct the input voltages to achieve the desired DICs.

Finally, one can imagine using the pipeline to hardcode target DIC values. Thus, the neurons would exhibit precise activity, and the chip could switch between patterns as needed. The practical benefits of this approach would vary depending on the application of the neuron outputs. In robotic applications, for instance, precise activity patterns could facilitate more accurate and refined control.

References

Almog, M., & Korngreen, A. (2016). Is realistic neuronal modeling realistic? *Journal of Neurophysiology*, *116*(5), 2180–2209. https://doi.org/10.1152/jn.00360.2016

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016, July). Layer Normalization. https://doi.org/10.48550/arXiv.1607.06450

Banach, S. (1922). Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae.*, *3*(1).

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, *18*(153), 1–43. http://jmlr.org/papers/v18/17-468.html

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, *13*(null), 281–305.

Berridge, M. J. (1998). Neuronal Calcium Signaling. *Neuron*, *21*(1), 13–26. https://doi.org/10.1016/S0896-6273(00)80510-3

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning* (1st ed.). Springer New York, NY.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2018, September). Conditional Time Series Forecasting with Convolutional Neural Networks. https://doi.org/10.48550/arXiv.1703.04691

Bucher, D., Taylor, A. L., & Marder, E. (2006). Central pattern generating neurons simultaneously express fast and slow rhythmic activities in the stomatogastric ganglion. *Journal of Neurophysiology*, *95*(6), 3617–3632. https://doi.org/10.1152/jn.00004.2006

Burden, R. L., & Faires, J. D. (2010, August). *Numerical Analysis*. Cengage Learning.

Burghi, T. B., O'Leary, T., & Sepulchre, R. (2022). Distributed online estimation of biophysical neural networks. *2022 IEEE 61st Conference on Decision and Control (CDC)*, 628–634. https://doi.org/10.1109/CDC51059.2022.9993018

Burghi, T. B., Schoukens, M., & Sepulchre, R. (2021). Feedback identification of conductance-based models. *Automatica*, *123*, 109297. https://doi.org/10.1016/j.automatica.2020.109297

Cranmer, K., Brehmer, J., & Louppe, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, *117*(48), 30055–30062. https://doi.org/10.1073/pnas.1912789117

Dayan, P., & Abbott, L. (2001). Theoretical neuroscience: Computational and mathematical modeling of neural systems [Chapter 1.4: Poisson Processes]. MIT Press.

Ditlevsen, S., & Samson, A. (2014). Estimation in the partially observed stochastic Morris–Lecar neuronal model with particle filter and stochastic approximation methods. *The Annals of Applied Statistics*, *8*(2), 674–702. https://doi.org/10.1214/14-AOAS729

Drion, G., Franci, A., Dethier, J., & Sepulchre, R. (2015). Dynamic input conductances shape neuronal spiking. *eNeuro*, *2*(1), ENEURO.0031–14.2015. https://doi.org/10.1523/ENEURO.0031-14.2015

Druckmann, S., Banitt, Y., Gidon, A. A., Schürmann, F., Markram, H., & Segev, I. (2007). A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data. *Frontiers in Neuroscience*, *1*. https://doi.org/10.3389/neuro.01.1.1.001.2007

Engel, T. A., Schimansky-Geier, L., Herz, A., Schreiber, S., & Erchova, I. (2008). Subthreshold Membrane-Potential Resonances Shape Spike-Train Patterns in the Entorhinal Cortex. *Journal of Neurophysiology*, *100*(3), 1576–1589. https://doi.org/10.1152/jn.01282.2007

Fenichel, N. (1979). Geometric singular perturbation theory for ordinary differential equations. *Journal of Differential Equations*, *31*(1), 53–98. https://doi.org/10.1016/0022-0396(79)90152-9

Franci, A., Drion, G., & Sepulchre, R. (2014). Modeling the modulation of neuronal bursting: A singularity theory approach. *SIAM Journal on Applied Dynamical Systems*, *13*(2), 798–829. https://doi.org/10.1137/13092263X

Franci, A., Drion, G., & Sepulchre, R. (2018). Robust and tunable bursting requires slow positive feedback. *Journal of Neurophysiology*, *119*(3), 1222–1234. https://doi.org/10.1152/jn.00804.2017

Franci, A., Drion, G., Seutin, V., & Sepulchre, R. (2013). A balance equation determines a switch in neuronal excitability. *PLOS Computational Biology*, *9*(5), 1–16. https://doi.org/10.1371/journal.pcbi.1003040

Fyon, A., Sacré, P., Franci, A., & Drion, G. (2023). Reliable neuromodulation from adaptive control of ion channel expression. *IFAC-PapersOnLine*, *56*(2), 458–463. https://doi.org/10.1016/j.ifacol.2023.10.1610

Fyon, A., & Drion, G. (2024). Neuromodulation and homeostasis: Complementary mechanisms for robust neural function. https://arxiv.org/abs/2412.04172

Fyon, A., Franci, A., Sacré, P., & Drion, G. (2024). Dimensionality reduction of neuronal degeneracy reveals two interfering physiological mechanisms. *PNAS Nexus*. https://doi.org/10.1093/pnasnexus/pgae415

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., & Teh, Y. W. (2018, July). Neural Processes. https://doi.org/10.48550/arXiv.1807.01622

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256. https://proceedings.mlr.press/v9/glorot10a.html

Goaillard, J.-M., & Marder, E. (2021). Ion channel degeneracy, variability, and covariation in neuron and circuit resilience. *Annual Review of Neuroscience*, *44*, 335–357. https://doi.org/10.1146/annurev-neuro-092920-121538

Goldwyn, J. H., & Shea-Brown, E. (2011). The What and Where of Adding Channel Noise to the Hodgkin-Huxley Equations. *PLOS Computational Biology*, *7*(11), e1002247. https://doi.org/10.1371/journal.pcbi.1002247

Golowasch, J., Goldman, M. S., Abbott, L. F., & Marder, E. (2002). Failure of Averaging in the Construction of a Conductance-Based Neuron Model. *Journal of Neurophysiology*, *87*(2), 1129–1131. https://doi.org/10.1152/jn.00412.2001

Gonçalves, P. J., Lueckmann, J.-M., Deistler, M., Nonnenmacher, M., Öcal, K., Bassetto, G., Chintaluri, C., Podlaski, W. F., Haddad, S. A., Vogels, T. P., Greenberg, D. S., & Macke, J. H. (2020). Training deep neural density estimators to identify mechanistic models of neural dynamics. *eLife*, *9*, e56261. https://doi.org/10.7554/eLife.56261

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Hammond, C. (2024). *Cellular and molecular neurophysiology* (5th). Elsevier Science & Technology.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Heiligenberg, W. (1991). *Neural nets in electric fish*. MIT Press.

Hendrycks, D., & Gimpel, K. (2023, June). Gaussian Error Linear Units (GELUs). https://doi.org/10.48550/arXiv.1606.08415

Hille, B. (1984). *Ionic channels of excitable membranes*. Sinauer Associates Inc.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012, July). Improving neural networks by preventing co-adaptation of feature detectors. https://doi.org/10.48550/arXiv.1207.0580

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544. https://doi.org/https://doi.org/10.1113/jphysiol.1952.sp004764

Houghton, C., & Kreuz, T. (2012). On the efficient calculation of van Rossum distances. *Network (Bristol, England)*, *23*(1-2), 48–58. https://doi.org/10.3109/0954898X.2012.673048

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021, October). LoRA: Low-Rank Adaptation of Large Language Models. https://doi.org/10.48550/arXiv.2106.09685

Huys, Q. J. M., Ahrens, M. B., & Paninski, L. (2006). Efficient Estimation of Detailed Single-Neuron Models. *Journal of Neurophysiology*, *96*(2), 872–890. https://doi.org/10.1152/jn.00079.2006

Huys, Q. J. M., & Paninski, L. (2009). Smoothing of, and Parameter Estimation from, Noisy Biophysical Recordings. *PLOS Computational Biology*, *5*(5), e1000379. https://doi.org/10.1371/journal.pcbi.1000379

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., van Schaik, A., Etienne-Cummings, R., Delbruck, T., Liu, S.-C., Dudek, P., Häfliger, P., Renaud, S., Schemmel, J., Cauwenberghs, G., Arthur, J., Hynna, K., Folowosele, F., Saïghi, S., Serrano-Gotarredona, T., Wijekoon, J., Wang, Y., & Boahen, K. (2011). Neuromorphic Silicon Neuron Circuits. *Frontiers in Neuroscience*, *5*. https://doi.org/10.3389/fnins.2011.00073

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning*, 448–456. https://proceedings.mlr.press/v37/ioffe15.html

Izhikevich, E. M. (2001). Resonate-and-fire neurons. *Neural Networks: The Official Journal of the International Neural Network Society*, *14*(6-7), 883–894.

Izhikevich, E. M. (2007). *Dynamical systems in neuroscience: The geometry of excitability and bursting*. MIT Press.

Johnson, K. O., & Hsiao, S. S. (1992). Neural Mechanisms of Tactual form and Texture Perception. *Annual Review of Neuroscience*, *15*(Volume 15, 1992), 227–250. https://doi.org/10.1146/annurev.ne.15.030192.001303

Kaufman, S., Rosset, S., Perlich, C., & Stitelman, O. (2012). Leakage in data mining: Formulation, detection, and avoidance. *ACM Trans. Knowl. Discov. Data*, *6*(4), 15:1–15:21. https://doi.org/10.1145/2382577.2382579

Kingma, D. P., & Ba, J. (2017, January). Adam: A Method for Stochastic Optimization. https://doi.org/10.48550/arXiv.1412.6980

Kudithipudi, D., Schuman, C., Vineyard, C. M., Pandit, T., Merkel, C., Kubendran, R., Aimone, J. B., Orchard, G., Mayr, C., Benosman, R., Hays, J., Young, C., Bartolozzi, C., Majumdar, A., Cardwell, S. G., Payvand, M., Buckley, S., Kulkarni, S., Gonzalez, H. A., . . . Furber, S. (2025). Neuromorphic computing at scale. *Nature*, *637*(8047), 801–812. https://doi.org/10.1038/s41586-024-08253-8

Lampl, I., & Yarom, Y. (1993). Subthreshold oscillations of the membrane potential: A functional synchronizing and timing device. *Journal of Neurophysiology*, *70*(5), 2181–2186. https://doi.org/10.1152/jn.1993.70.5.2181

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient BackProp. In *Neural Networks: Tricks of the Trade: Second Edition* (pp. 9–48). Springer. https://doi.org/10.1007/978-3-642-35289-8_3

Liu, Z., Golowasch, J., Marder, E., & Abbott, L. F. (1998). A Model Neuron with Activity-Dependent Conductances Regulated by Multiple Calcium Sensors. *Journal of Neuroscience*, *18*(7), 2309–2320. https://doi.org/10.1523/JNEUROSCI.18-07-02309.1998

Loshchilov, I., & Hutter, F. (2019, January). Decoupled Weight Decay Regularization. https://doi.org/10.48550/arXiv.1711.05101

Louppe, G. (2023). Info8010 - deep learning [Version 7 (INFO8010, ULiège, 2023-2024)]. https://github.com/glouppe/info8010-deep-learning

Louppe, G., Wehenkel, L., Sutera, A., & Geurts, P. (2013). Understanding variable importances in forests of randomized trees. *Advances in Neural Information Processing Systems*, *26*.

Mainen, Z. F., & Sejnowski, T. J. (1995). Reliability of spike timing in neocortical neurons. *Science (New York, N.Y.)*, *268*(5216), 1503–1506. https://doi.org/10.1126/science.7770778

Marder, E. (2011). Variability, compensation, and modulation in neurons and circuits. *Proceedings of the National Academy of Sciences*, *108*(supplement_3), 15542–15548. https://doi.org/10.1073/pnas.1010674108

Marder, E., & Taylor, A. L. (2011). Multiple models to capture the variability in biological neurons and networks. *Nature Neuroscience*, *14*(2), 133–138. https://doi.org/10.1038/nn.2735

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, *21*(2), 239–245. https://doi.org/10.2307/1268522

Mendolia, L., Wen, C., Chicca, E., Indiveri, G., Redouté, J.-M., & Franci, A. (2025). *A neuromodulable current-mode silicon neuron for robust and adaptive neuromorphic systems* [In preparation].

Meng, L., Kramer, M. A., & Eden, U. T. (2011). A sequential Monte Carlo approach to estimate biophysical neural models from spikes. *Journal of Neural Engineering*, *8*(6), 065006. https://doi.org/10.1088/1741-2560/8/6/065006

Meng, L., Kramer, M. A., Middleton, S. J., Whittington, M. A., & Eden, U. T. (2014). A Unified Approach to Linking Experimental, Statistical and Computational Analysis of Spike Train Data. *PLOS ONE*, *9*(1), e85269. https://doi.org/10.1371/journal.pone.0085269

Nadim, F., Olsen, O. H., De Schutter, E., & Calabrese, R. L. (1995). Modeling the leech heartbeat elemental oscillator. I. Interactions of intrinsic and synaptic currents. *Journal of Computational Neuroscience*, *2*(3), 215–235. https://doi.org/10.1007/BF00961435

Naudin, L. (2023). Different parameter solutions of a conductance-based model that behave identically are not necessarily degenerate. *Journal of Computational Neuroscience*, *51*(2), 201–206. https://doi.org/10.1007/s10827-023-00848-w

Nobel Prize Outreach. (1963). The nobel prize in physiology or medicine 1963 [Accessed: 24-Feb-2025]. https://www.nobelprize.org/prizes/medicine/1963/summary/

Nobel Prize Outreach. (2024). The nobel prize in physics 2024 [Accessed: 31-Mar-2025]. https://www.nobelprize.org/prizes/physics/2024/summary/

O'Leary, T., Williams, A. H., Franci, A., & Marder, E. (2014). Cell types, network homeostasis, and pathological compensation from a biologically plausible ion channel expression model. *Neuron*, *82*(4), 809–821. https://doi.org/10.1016/j.neuron.2014.04.002

Paninski, L., Pillow, J. W., & Simoncelli, E. P. (2004). Maximum likelihood estimation of a stochastic integrate-and-fire neural encoding model. *Neural Computation*, *16*(12), 2533–2561. https://doi.org/10.1162/0899766042321797

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch. *NIPS-W*.

Petneházi, G. (2019, January). Recurrent Neural Networks for Time Series Forecasting. https://doi.org/10.48550/arXiv.1901.00069

Prinz, A. A., Billimoria, C. P., & Marder, E. (2003). Alternative to hand-tuning conductance-based models: Construction and analysis of databases of model neurons. *Journal of Neurophysiology*, *90*(6), 3998–4015. https://doi.org/10.1152/jn.00641.2003

Qian, K., Yu, N., Tucker, K. R., Levitan, E. S., & Canavier, C. C. (2014). Mathematical analysis of depolarization block mediated by slow inactivation of fast sodium channels in midbrain dopamine neurons. *Journal of Neurophysiology*, *112*(11), 2779–2790. https://doi.org/10.1152/jn.00578.2014

Rossant, C., Goodman, D. F. M., Fontaine, B., Platkiewicz, J., Magnusson, A. K., & Brette, R. (2011). Fitting Neuron Models to Spike Trains. *Frontiers in Neuroscience*, *5*. https://doi.org/10.3389/fnins.2011.00009

Rossum, M. C. W. v. (2001). A Novel Spike Distance. *Neural Computation*, *13*(4), 751–763. https://doi.org/10.1162/089976601300014321

Saghafi, S., Rumbell, T., Gurev, V., Kozloski, J., Tamagnini, F., Wedgwood, K. C. A., & Diekman, C. O. (2024). Inferring Parameters of Pyramidal Neuron Excitability in Mouse Models of Alzheimer's Disease Using Biophysical Modeling and Deep Learning. *Bulletin of Mathematical Biology*, *86*(5), 46. https://doi.org/10.1007/s11538-024-01273-5

Seidemann, E., Meilijson, I., Abeles, M., Bergman, H., & Vaadia, E. (1996). Simultaneously recorded single units in the frontal cortex go through sequences of discrete and stable states in monkeys performing a delayed localization task. *Journal of Neuroscience*, *16*(2), 752–768. https://doi.org/10.1523/JNEUROSCI.16-02-00752.1996

Sepulchre, R., Drion, G., & Franci, A. (2019). Control across scales by positive and negative feedback. *Annual Review of Control, Robotics, and Autonomous Systems*, *2*(Volume 2), 89–113. https://doi.org/10.1146/annurev-control-053018-023708

Shampine, L. F., & Reichelt, M. W. (1997). The MATLAB ODE Suite. *SIAM Journal on Scientific Computing*, *18*(1), 1–22. https://doi.org/10.1137/S1064827594276424

Sharma, D., Ng, K. K. W., Birznieks, I., & Vickery, R. M. (2022). The burst gap is a peripheral temporal code for pitch perception that is shared across audition and touch. *Scientific Reports*, *12*(1), 11014. https://doi.org/10.1038/s41598-022-15269-5

Shepardson, D. (2009, December). *Algorithms for Inverting Hodgkin-Huxley Type Neuron Models* [Ph.D. Thesis]. Georgia Institute of Technology. http://hdl.handle.net/1853/31686

Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., & Raffel, C. (2022, August). Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning. https://doi.org/10.48550/arXiv.2205.05638

Taylor, A. L., Goaillard, J.-M., & Marder, E. (2009). How multiple conductances determine electrophysiological properties in a multicompartment model. *Journal of Neuroscience*, *29*(17), 5573–5586. https://doi.org/10.1523/JNEUROSCI.4438-08.2009

Temporal, S., Desai, M., Khorkova, O., Varghese, G., Dai, A., Schulz, D. J., & Golowasch, J. (2012). Neuromodulation independently determines correlated channel expression and conductance levels in motor neurons of the stomatogastric ganglion. *Journal of Neurophysiology*, *107*(2), 718–727. https://doi.org/10.1152/jn.00622.2011

Traub, R. D., Wong, R. K., Miles, R., & Michelson, H. (1991). A model of a CA3 hippocampal pyramidal neuron incorporating voltage-clamp data on intrinsic conductances. *Journal of Neurophysiology*, *66*(2), 635–650. https://doi.org/10.1152/jn.1991.66.2.635

Turrigiano, G., LeMasson, G., & Marder, E. (1995). Selective regulation of current densities underlies spontaneous changes in the activity of cultured neurons. *Journal of Neuroscience*, *15*(5), 3640–3652. https://doi.org/10.1523/JNEUROSCI.15-05-03640.1995

Valero, M., Zutshi, I., Yoon, E., & Buzsáki, G. (2022). Probing subthreshold dynamics of hippocampal neurons by pulsed optogenetics. *Science (New York, N.Y.)*, *375*(6580), 570–574. https://doi.org/10.1126/science.abm1891

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023, August). Attention Is All You Need. https://doi.org/10.48550/arXiv.1706.03762

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., . . . SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wang, Z., Ying, Z., Bosy-Westphal, A., Zhang, J., Schautz, B., Later, W., Heymsfield, S. B., & Müller, M. J. (2010). Specific metabolic rates of major organs and tissues across adulthood. *The American Journal of Clinical Nutrition*, *92*(6), 1369–1377. https://doi.org/10.3945/ajcn.2010.29885

Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., & Sun, L. (2023, May). Transformers in Time Series: A Survey. https://doi.org/10.48550/arXiv.2202.07125

Yang, Z., Zhao, Q., Keefer, E., & Liu, W. (2009). Noise Characterization, Modeling, and Reduction for In Vivo Neural Recording. *Advances in Neural Information Processing Systems*, *22*.

Yu, N., & Canavier, C. C. (2015). A mathematical model of a midbrain dopamine neuron identifies two slow variables likely responsible for bursts evoked by sk channel antagonists and terminated by depolarization block. *The Journal of Mathematical Neuroscience (JMN)*, *5*(1), 5. https://doi.org/10.1186/s13408-015-0017-6

Outline of the Appendices

The remainder of this document comprises appendices that provide supplementary material supporting the main content of the thesis. Each appendix offers detailed information on specific topics referenced in the main chapters.

Structure of the Appendices

The first appendix, **A About the Use of Artificial Intelligent Conversational Agents**, discusses the utilization of AI tools such as ChatGPT, Le Chat, and GitHub Copilot in the preparation of the thesis, emphasizing transparency.

The second appendix, **B Numerical Details on Conductance-Based Models and Their Simulation**, provides detailed equations and numerical resolution techniques for the CBMs used in the study.

The third appendix, **C More about Dynamic Input Conductances**, offers additional insights into the DICs theory, including differences from the original formulation and numerical procedures for reachability analysis.

The fourth appendix, **D Neuronal Activity Metrics and Comparison Methods**, formalizes the definitions of quantitative metrics used to describe neuronal activity and introduces the van Rossum distance for comparing activity differences.

The fifth appendix, **E Pre-Generation Analyses and Rationale**, details the analyses and rationale behind the choices made prior to the generation of degenerate populations of CBMs, including conductance distributions and threshold potential values.

The sixth appendix, **F The AdamW Optimizer and Training Details**, provides an overview of the AdamW optimizer, its configuration, and the implementation details of the training process for the models, including the use of the PyTorch framework.

The seventh appendix, **G Transfer Details**, summarizes the details related to the transfer of the method from the STG neuron model to the DA neuron model.

Appendix A

About the Use of Artificial Intelligent Conversational Agents

In the preparation of this thesis, I have utilized several artificial intelligent conversational agents, namely ChatGPT, Le Chat, and GitHub Copilot. This section serves as a declaration of transparency and an honor statement affirming that the use of these tools has been limited to the purposes described herein.

ChatGPT and GitHub Copilot were primarily used for code generation. The code was not generated from scratch but rather built upon existing codebases that I personally developed (or translated from Julia into Python based on Arthur Fyon's work[1]). I requested vectorized versions of the code using NumPy for computationally intensive operations or parallelized codes, particularly for the code related to simulations. They were also used for graphical rendering in Python before I personally reworked the figures in Inkscape.

Le Chat was the only tool that had a direct impact on the text within the manuscript. It is important to emphasize that it did not write the sections for me from nothing. In practice, it assisted in drafting academic English and verifying syntax. I provided it with the following prompt: *"You are an expert in the field. Please help me with my thesis. You should translate into English if needed, review syntax, spelling, and the flow of ideas but don't change the general idea. [. . . part of my text . . .]."* In summary, the ideas presented in this thesis are derived from the cited references, my personal insights, and discussions I have had with my supervisor, co-supervisor, fellow Master's students, and other doctoral candidates in the laboratory. Not from any artificial agent; this work is truly a personal endeavor.

[1]The implementation of the DIC calculations is primarily based on the following repository: https://github.com/arthur-fyon/CORR_2024. While the Python translation is optimized using NumPy vectorized operations, the core ideas for this part originate from this source. The remainder of the work, including model simulations, graphical representations, applications, and the PyTorch implementation, is original and developed independently.

Appendix B

Numerical Details on Conductance-Based Models and Their Simulation

This appendix aims to detail the equations of the Conductance-Based Models (CBMs) used in this work, as well as the details related to their simulation — that is, the numerical resolution of the associated system of ODEs. We first detail the two models and then, in a common section, their simulation.

B.1 About the STG Model

The STG model includes eight ionic currents, including the leak current. Table B.5 lists the mathematical expressions for the steady-state gating functions (Fig. B.1), the time constants (Fig. B.2), and the corresponding gating exponents. Table B.1 provides the values of the fixed model parameters. These expressions and parameters follow the formulation described in Box III. The calcium dynamics are governed by Eq. (2.5). The membrane capacitance is set to $1\,\mu\mathrm{F\,cm^{-2}}$.

E_{leak}	E_{Na}	E_{K}	E_{H}	E_{Ca}	τ_{Ca}	α_{Ca}	β_{Ca}
$-50\,\mathrm{mV}$	$50\,\mathrm{mV}$	$-80\,\mathrm{mV}$	$-20\,\mathrm{mV}$	$80\,\mathrm{mV}$	$20\,\mathrm{ms}$	$0.94\,\mathrm{mM\,nF\,nA^{-1}}$	$0.05\,\mu\mathrm{M}$

Table B.1: Fixed Parameters for the STG Model. Reversal potentials, calcium time constant, and calcium dynamics parameters used in the STG model.

V_0	Ca_0	$m_{i,0}$	$h_{i,0}$
$-70\,\mathrm{mV}$	$0.5\,\mu\mathrm{M}$	$m_{i,\infty}(V_0)$ or $m_{i,\infty}(V_0, \text{Ca}_0)$	$h_{i,\infty}(V_0)$

Table B.2: Initial Conditions for the STG Model. Initial values for the membrane potential, calcium concentration, and gating variables used in simulation.

Differences between the implementation in this work and that of Liu et al., 1998 While the core structure of the presented model remains unchanged, it is important to highlight two differences to ensure the reproducibility of results:

1. **Simplification of calcium influence**. In the 1998 model, to better align with biological reality, the Nernst potential of calcium, E_{Ca}, evolves dynamically as a function of Ca, following the Nernst equation (2.1). In the implementation used in this work, however, E_{Ca} is fixed, independent of calcium dynamics.

   This simplification was chosen to facilitate the application of the DICs theory introduced in Section 2.3. It has however some small impact on the observed behavior for a given instance of the model $\bar{\boldsymbol{g}}_{\text{STG}}$. With an appropriate treatment of the theory, this minor approximation can be accounted for.

2. **Activity-<u>in</u>dependent maximal conductances**. The primary objective of Liu et al., 1998 was to propose a model where maximal conductances are influenced by calcium dynamics. In contrast, this work assumes $\dot{\bar{g}}_i = 0$, meaning that maximal conductances are treated as constant.

This simplification is not problematic in itself, as we consider only the steady-state behavior of the system. While Liu et al., 1998 models activity-dependent regulation of conductances, one can also interpret their evolving conductances as a sequence of steady-state configurations.

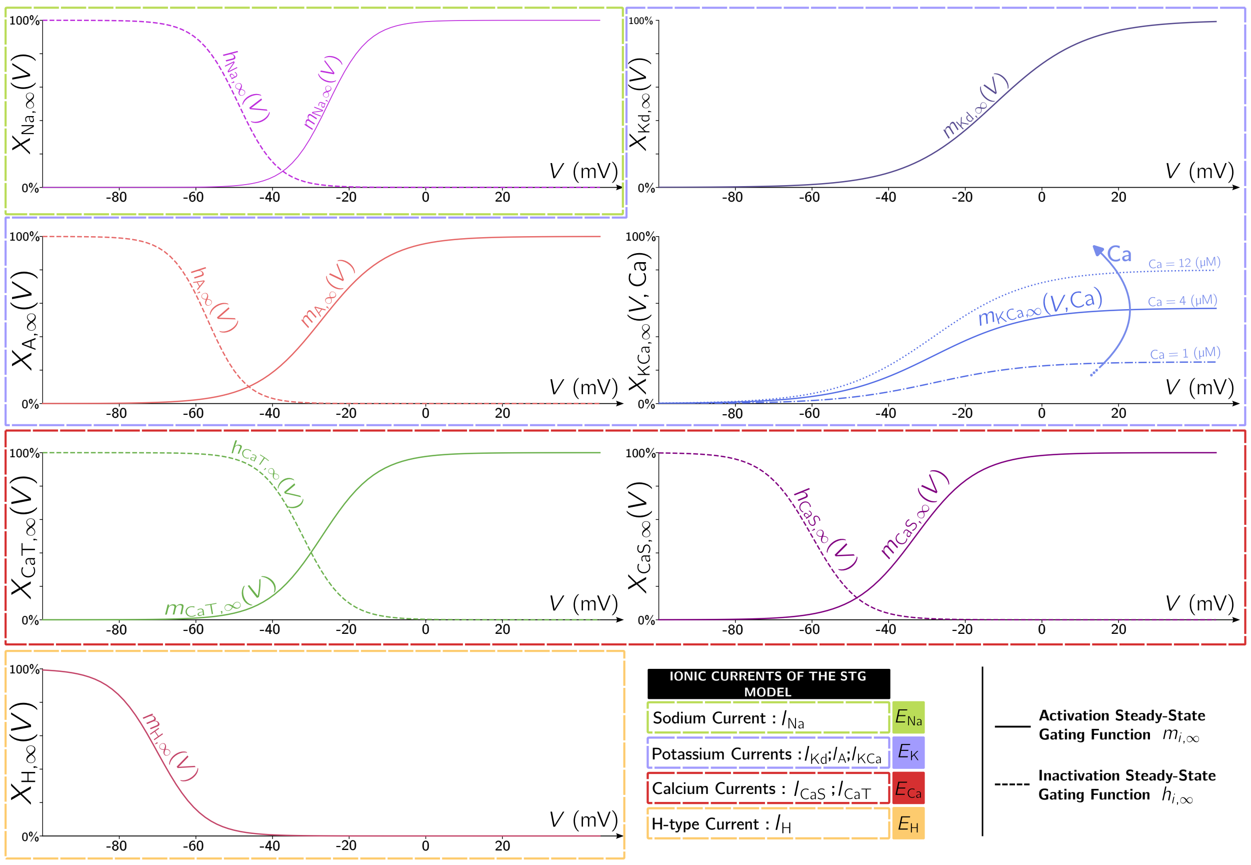


Figure B.1: Voltage-dependent steady-state gating functions of the STG model. Each subplot displays the steady-state activation (m_∞, solid lines) and/or inactivation (h_∞, dashed lines) gating variables as functions of membrane potential V for different ionic currents in the STG model. The first row shows the gating functions for the sodium current (I_{Na}, left) and the delayed-rectifier potassium current (I_{Kd}, right). The second row presents the A-type potassium current (I_A, left) and the calcium-dependent potassium current (I_{KCa}, right). The third row includes the transient (I_{CaT}, left) and slow (I_{CaS}, right) calcium currents. The fourth row illustrates the hyperpolarization-activated cation current (I_H, left). The calcium-dependent activation function $m_{KCa,\infty}(V, Ca)$ is plotted for different intracellular calcium concentrations ($Ca = 1, 4, 12\,\mu M$). Currents associated with the same ion share the same Nernst potential: potassium currents (I_{Kd}, I_{KCa}, I_A) have E_K, calcium currents (I_{CaT}, I_{CaS}) have E_{Ca}, sodium current (I_{Na}) has E_{Na}, and the H-type current (I_H) has E_H. The full model also includes a leak current associated with the leak reversal potential E_{leak}.

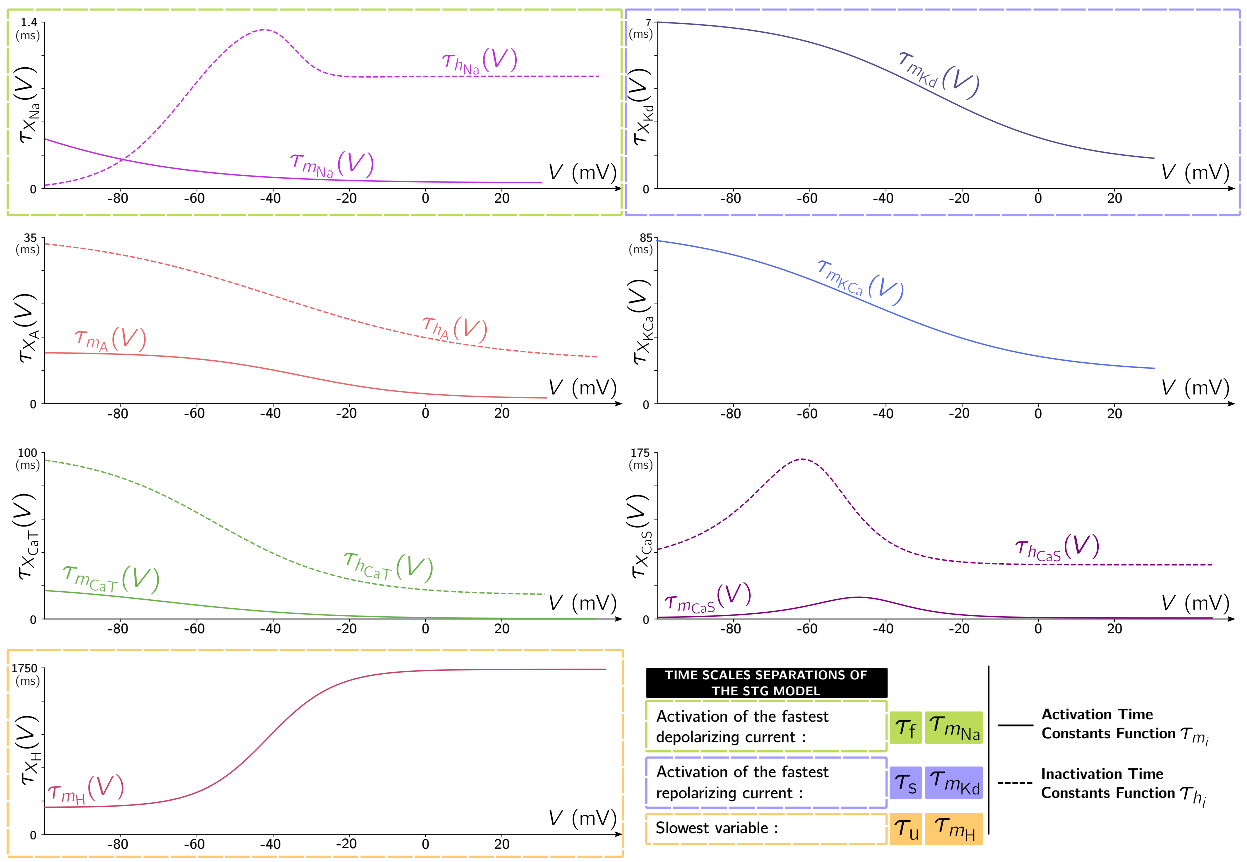


Figure B.2: Voltage-dependent time constants of the STG model. Each subplot displays the voltage-dependent time constant $\tau_{X_i}(V)$ (in ms) for the gating variables associated with different ionic currents in the STG model. These time constants determine the speed of activation (τ_{m_i}, solid lines) or inactivation (τ_{h_i}, dashed lines) of each current. The first row shows the time constants for the sodium current (I_{Na}, left) and the delayed-rectifier potassium current (I_{Kd}, right). The second row presents the A-type potassium current (I_A, left) and the calcium-dependent potassium current (I_{KCa}, right). The third row includes the transient (I_{CaT}, left) and slow (I_{CaS}, right) calcium currents. The fourth row illustrates the hyperpolarization-activated cation current (I_H, left). In a timescale separation, the fast reference is $\tau_{m_{Na}}(V)$, the slow reference is $\tau_{m_{Kd}}(V)$, and the ultra-slow reference is $\tau_H(V)$.

B.2 About the DA Model

The DA model consists of six ionic currents in addition to the leak current, including Na, Kd, CaL, CaN, ERG (a potassium current), and NMDA (permeable to K^+, Na^+, and Ca^{2+}). Table B.6 details the gating functions, time constants, and exponents. Fixed model parameters are given in Table B.3, and initial simulation conditions in Table B.4. The capacitance is set to $1\,\mu F\,cm^{-2}$.

In this model, the NMDA current is considered instantaneous and always evaluated at its steady-state value:

$$I_{NMDA} = \bar{g}_{NMDA}(V - E_{NMDA}) \cdot m_{NMDA,\infty}(V, Mg)$$

where $m_{NMDA,\infty}$ is the voltage- and magnesium-dependent steady-state activation function.

For the ERG channel, the gating variables o_{ERG} and i_{ERG} evolve according to the following differential equations:

$$\frac{do_{ERG}}{dt} = a_0(V) \cdot (1 - o_{ERG} - i_{ERG}) + b_i(V) \cdot i_{ERG} - o_{ERG} \cdot (a_i(V) + b_0(V))$$

$$\frac{di_{ERG}}{dt} = a_i(V) \cdot o_{ERG} - b_i(V) \cdot i_{ERG}$$

The steady-state expressions for the ERG gating variables under constant voltage V are:

$$o_{\text{ERG},\infty}(V) = \frac{a_0(V) \cdot b_i(V)}{a_0(V)(a_i(V) + b_i(V)) + b_0(V) \cdot b_i(V)}$$

$$i_{\text{ERG},\infty}(V) = \frac{a_0(V) \cdot a_i(V)}{a_0(V)(a_i(V) + b_i(V)) + b_0(V) \cdot b_i(V)}$$

The corresponding ERG current is given by:

$$I_{\text{ERG}} = \bar{g}_{\text{ERG}} \cdot o_{\text{ERG}} \cdot (V - E_K)$$

E_{Na}	E_{K}	E_{Ca}	E_{leak}	E_{NMDA}	Mg
60 mV	−85 mV	60 mV	−50 mV	0 mV	1.4

Table B.3: Fixed Parameters for the DA Model. Reversal potentials and magnesium concentration used in the DA model.

V_0	$m_{i,0}$	$h_{i,0}$	x_0
−90 mV	$m_{i,\infty}(V_0)$	$h_{i,\infty}(V_0)$	$x_{i,\infty}(V_0)$

Table B.4: Initial Conditions for the DA Model. Initialization values used in simulations for the membrane potential and gating variables.

B.3 About the Numerical Resolution of CBMs

The simulations are conducted using Python 3.11.11 with the SciPy library (Virtanen et al., 2020). Specifically, the `solve_ivp` function with the BDF method (from Shampine and Reichelt, 1997) is employed. This method is particularly suitable for stiff systems of ODEs. Tests were also conducted using RK45, but BDF definitively yielded the best results. A stiff problem is characterized by different timescales, which is precisely the core of this work and thus applicable to neuron models. A maximum time step of $\Delta t = 0.05$ ms is used, corresponding to the sampling period for spike times, although the step size is adaptive within the method. Simulations are run for $T = 5000$ ms for the STG model and $T = 12\,000$ ms for the DA model. When simulating an instance, the observed activity at the beginning of the simulation is variable and strongly influenced by the initial conditions. After this transient activity, the instance exhibits its spontaneous, steady-state activity, which is of primary interest. In both cases, the first 3000 milliseconds are discarded to retain only the steady-state activity. For the STG model, the analytical expression of the Jacobian is provided to the BDF method. For the DA model, the Jacobian is approximated by the method. In practice, the CBM population simulation problem is embarrassingly parallel. Our implementation takes advantage of modern multi-core CPU architectures for faster generation.

$$f(V,A,B,C,D) = A + \frac{B}{1+\exp\left(\frac{V+D}{C}\right)} \tag{B.1}$$

Current I_i	p_i	q_i	$m_{i,\infty}(V)$ or $m_{i,\infty}(V,\text{Ca})$	$h_{i,\infty}(V)$	$\tau_{m_i}(V)$	$\tau_{h_i}(V)$
I_{Na}	3	1	$f(V,0,1,-5.29,25.5)$	$f(V,0,1,5.18,48.9)$	$f(V,1.32,-1.26,-25,120)$	$f(V,0,0.67,-10,62.9) \cdot f(V,1.5,1,3.6,34.9)$
I_{Kd}	4	0	$f(V,0,1,-11.8,12.3)$	—	$f(V,7.2,-6.4,-19.2,28.3)$	—
I_{CaT}	3	1	$f(V,0,1,-7.2,27.1)$	$f(V,0,1,5.5,32.1)$	$f(V,21.7,-21.3,-20.5,68.1)$	$f(V,105,-89.8,-16.9,55)$
I_{CaS}	3	1	$f(V,0,1,-8.1,33)$	$f(V,0,1,6.2,60)$	$1.4+\frac{7}{\exp\left(\frac{V+27}{10}\right)+\exp\left(\frac{V+70}{-13}\right)}$	$60+\frac{150}{\exp\left(\frac{V+55}{9}\right)+\exp\left(\frac{V+65}{-16}\right)}$
I_{KCa}	4	0	$\frac{\text{Ca}}{\text{Ca}+3} \cdot f(V,0,1,-12.6,28.3)$	—	$f(V,90.3,-75.1,-22.7,46)$	—
I_{A}	3	1	$f(V,0,1,-8.7,27.2)$	$f(V,0,1,4.9,56.9)$	$f(V,11.6,-10.4,-15.2,32.9)$	$f(V,38.6,-29.2,-26.5,38.9)$
I_{H}	1	0	$f(V,0,1,6,70)$	—	$f(V,272,1499,-8.73,42.2)$	—
I_{leak}	0	0	—	—	—	—

Table B.5: Gating Functions and Kinetics for the STG Model. Steady-state activation/inactivation functions, time constants, and exponents for each ionic current in the STG model. All f functions refer to the generalized sigmoid function (Eq. (B.1)). See also Figures B.1 and B.2.

Current	p_i	q_i	$m_{i,\infty}(V)$ or $m_{i,\infty}(V,\text{Mg})$	$h_{i,\infty}(V)$	$\tau_{m_i}(V)$	$\tau_{h_i}(V)$
I_{Na}	3	1	$f(V,0,1,-9.7264,30.0907)$	$f(V,0,1,10.7665,54.0289)$	$0.01+\frac{1.0}{\left(-\frac{15.6504+0.4043V}{\exp(-19.565-0.5052V)-1.0}\right)+3.0212\exp(-0.007463V)}$	$0.4+\frac{1.0}{(0.00050754\exp(-0.063213V))+9.7529\exp(0.13442V)}$
I_{Kd}	3	0	$f(V,0,1,-12,25)$	—	$f(V,20,-18,-10,38)$	—
I_{CaL}	2	0	$f(V,0,1,-2,50)$	—	$f(V,30,-28,-3,45)$	—
I_{CaN}	1	0	$f(V,0,1,-7,30)$	—	$f(V,30,-25,-6,55)$	—
I_{NMDA}	1	0	$\frac{1}{1+\frac{\text{Mg}\cdot\exp(-0.08V)}{10}}$	—	—	—
I_{leak}	0	0	—	—	—	—

Table B.6: Gating Functions and Kinetics for the DA Model. All f functions refer to the generalized sigmoid function (Eq. (B.1)). Additionally, there is a current I_{ERG} described by the ERG channel, whose rate functions are defined as exponentials of the membrane potential V. The activation and inactivation parameters are as follows: $a_0(V) = 0.0036\exp(0.0759V)$, $b_0(V) = 1.2523\times10^{-5}\exp(-0.0671V)$, $a_i(V) = 0.1\exp(0.1189V)$, and $b_i(V) = 0.003\exp(-0.0733V)$.

Appendix C

More about Dynamic Input Conductances

This appendix aims to provide more details about the theory of DICs. We outline the differences from the original formulation and detail the numerical procedures for reachability analysis and the implementation of the iterative compensation algorithm.

C.1 Differences from the Original Formulation

The DICs framework presented differs slightly from the version originally introduced in Drion et al., 2015. Those differences are aligned with more recent usage of DICs (e.g., Fyon et al., 2024). The three main differences are:

- **Sign inversion**: The sign of the DICs in this work is reversed compared to Drion et al., 2015. This choice does not affect the fundamental conceptualization of the tool but is a matter of interpretation. As explained, neurons can be analyzed as mixed-feedback systems, where it is more intuitive to use the original Drion et al., 2015 definition — where a positive DIC indicates positive feedback, and a negative DIC indicates negative feedback. However, in the context of CBM analysis, where biological parallels are emphasized, the opposite sign convention is more intuitive. Under this convention, DICs behave more like true *conductances*: a positive DIC corresponds to a stabilizing effect (outward positive current when voltage increases), while a negative DIC corresponds to a reinforcing effect (inward positive current when voltage increases).

- **Inclusion of passive membrane behavior**: The original DICs formulation neglected the passive properties of the membrane, considering them instantaneous. More recent studies have included this behavior, which is also the approach taken in this work. This is reflected in the term $\frac{\partial \dot{V}}{\partial V}$, which acts only on the fast timescale in equations (2.11). This term represents the *instantaneous* movement of ions across the membrane and accounts for effects such as the leak current.

- **Normalization**: In this work, we introduce a normalization factor by g_{leak} in the definition of the DICs. This factor allows us to obtain dimensionless DICs and control the effect of *homogeneous scaling* presented by Fyon et al., 2024, which are described in detail in Section 2.4.

Additionally, one more modification is introduced:

- **Ultra-slow dynamics in the STG model**: In the 2015 paper, the timescale separation for the STG model was based on $\tau_{\text{CaS}}(V)$ as the slowest variable in the system. In this work, the I_{H} current is included, and $\tau_{m_{\text{H}}}(V)$ is used as the reference for the ultra-slow timescale $\tau_{\text{us}}(V)$.

These differences explain why Figure 2.10 has a slightly different shape (flipped along the x-axis with minor distortions) compared to Drion et al., 2015, Figure 3.

C.2 Implementation of the Iterative Generation Algorithm

The iterative generation procedure introduced as a major contribution of this work can be described in pseudocode for easy implementation and understanding. In practice, we modify Algorithm 1, originally introduced by Fyon et al., and incorporate iterative steps (Algorithm 3). The final result is Algorithm 4.

C.3 The Reachability Analysis

Reachability maps can be obtained through a simple and highly parallelizable numerical procedure. Algorithm 2 summarizes the procedure in pseudocode.

In brief, the procedure consists of discretizing the DICs space into a finite number of points. In the case of the STG model, the grid $(g_\mathrm{s}, g_\mathrm{u}) \in [-20, 20] \times [0, 20]$ is evenly sampled with a step size of $\delta g = 0.5$. For each point in the DICs space, we use the generation procedure to obtain M instances — here, $M = 1000$ instances are used. We then filter the resulting population by removing (1) instances with one or more negative conductances, and (2) instances whose DIC values do not match the target values, by using a threshold on the Euclidean distance: $\left\|(g_\mathrm{s}, g_\mathrm{u})_i - \boldsymbol{g}_{\mathrm{DICs},i}\right\| \leq \epsilon$ — where $\epsilon = 0.1$ is used, and the number of iterations of the generation procedure is increased to 25.

Algorithm 2: Numerical Procedure for Reachability Analysis

```
Data:
    - The size of the populations: $M$
    - DIC ranges: $g_\mathrm{s} \in [g_\mathrm{s,min}, g_\mathrm{s,max}]$, $g_\mathrm{u} \in [g_\mathrm{u,min}, g_\mathrm{u,max}]$
    - Step size for discretization: $\delta g$
    - Conductances to be compensated: $\bar{\boldsymbol{g}}_\mathrm{comp.}$
    - DICs distance threshold: $\epsilon$
Result: Reachability map indicating the probability of successful compensation for each point in the DICs space.
begin
    // Discretize the DICs space
    Construct a uniform grid $(g_\mathrm{s}, g_\mathrm{u})$ with step size $\delta g$
    // Initialize the reachability grid
    Create an empty 2D array 'reachability_grid' of size $(\mathrm{len}(g_\mathrm{s}), \mathrm{len}(g_\mathrm{u}))$
    foreach $(g_s, g_u)$ in the grid do
        // Generate a population with the target DICs
        Sample $M$ individuals using Algorithm 4 with $\bar{\boldsymbol{g}}_\mathrm{comp.}$
        // Filter out invalid instances
        Remove individuals with any negative conductance values
        Compute residuals $\|(g_\mathrm{s}, g_\mathrm{u}) - \boldsymbol{g}_\mathrm{DICs}\|$
        Remove individuals where residual $> \epsilon$
        // Compute reachability percentage
        reachability$(g_\mathrm{s}, g_\mathrm{u}) \leftarrow \frac{\text{\# remaining individuals}}{M}$
        // Store the reachability value in the grid
        Update 'reachability_grid[$(g_\mathrm{s}, g_\mathrm{u})$]' with the calculated reachability
    return Reachability grid 'reachability_grid'
```

Algorithm 3: Iterative Compensation Step

```
Input:
    - Target values $\boldsymbol{b}$
    - Initial approximation $\boldsymbol{x}_0$ or target-dependent model $\hat{\boldsymbol{f}}(\boldsymbol{b})$
    - Number of iterations $K$
    - Function $\boldsymbol{A}(\boldsymbol{x})$ that defines the system to solve: $\boldsymbol{A}(\boldsymbol{x})\boldsymbol{x} = \boldsymbol{b}$
Output: Approximate solution $\boldsymbol{x}_K$
begin
    Step 1: Initialize the process
    if using target-dependent model then
        $\boldsymbol{x}_0 \leftarrow \hat{\boldsymbol{f}}(\boldsymbol{b})$          // Compute initial estimate
    Step 2: Iterative Compensation
    for $k = 0, 1, \ldots, K-1$ do
        // 1. Compute system matrix using current estimate:
        $\boldsymbol{A}_k^* = \boldsymbol{A}(\boldsymbol{x}_k)$
        // 2. Solve linear system for next iterate:
        $\boldsymbol{x}_{k+1} = (\boldsymbol{A}_k^*)^{-1}\boldsymbol{b}$
    Return $\boldsymbol{x}_K$
```

Algorithm 4: Efficient *Iterative* Generation of Degenerate Populations with Spontaneous Activity

```
Data:
    - The size of the population: $M$
    - Conductance distribution $\mathcal{D}$: $\bar{\boldsymbol{g}}_{\text{random}} \sim \mathcal{D}$
    - Threshold potential $V_{\text{th}}$
    - Target DIC values at threshold: $\boldsymbol{g}_{\text{DICs}}$
    - Conductances to be compensated: $\bar{\boldsymbol{g}}_{\text{comp.}} \subseteq \bar{\boldsymbol{g}}$
    - Target DIC values for spontaneous spiking: $\boldsymbol{g}_{\text{DICs, spont.}}$
    - Conductances to be compensated for spontaneous spiking: $\bar{\boldsymbol{g}}_{\text{comp., spont.}} \subseteq \bar{\boldsymbol{g}}$
Result: A degenerate population exhibiting spontaneous activity
begin
    for $i \leftarrow 1$ to $M$ do
        // Step 1: Generate an individual with spontaneous spiking
        Sample $\bar{\boldsymbol{g}}_i \sim \mathcal{D}$
        // Construct sensitivity matrices
        Compute $\boldsymbol{S}(V_{\text{th}})$, the sensitivity matrix associated with DICs at threshold
        Extract $\boldsymbol{S}_{\text{comp., spont.}}$, the submatrix corresponding to compensated conductances
        Extract $\boldsymbol{S}_{\text{random}}$, the submatrix corresponding to randomly sampled conductances
        // Adjust conductances for spontaneous spiking
        if Nonlinear compensation is detected then
            $\bar{\boldsymbol{g}}_{\text{comp., spont.}} \leftarrow$ Iterative Compensation Step($\boldsymbol{b} = \boldsymbol{g}_{\text{DICs, spont.}}$, $\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{S}(\bar{\boldsymbol{g}})$)
        else
            Solve the system:
                $\boldsymbol{S}_{\text{comp., spont.}} \cdot \bar{\boldsymbol{g}}_{\text{comp., spont.}} = \boldsymbol{g}_{\text{DICs, spont.}} - \boldsymbol{S}_{\text{random}} \cdot \bar{\boldsymbol{g}}_{\text{random}}$
        // Step 2: Modulate the individual to refine spontaneous activity
        // Construct sensitivity matrices for this step
        Extract $\boldsymbol{S}_{\text{comp.}}$ and $\boldsymbol{S}_{\text{random}}$
        // Adjust conductances for target DIC values
        if Nonlinear compensation is detected then
            $\bar{\boldsymbol{g}}_{\text{comp.}} \leftarrow$ Iterative Compensation Step($\boldsymbol{b} = \boldsymbol{g}_{\text{DICs}}$, $\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{S}(\bar{\boldsymbol{g}})$)
        else
            Solve the system:
                $\boldsymbol{S}_{\text{comp.}} \cdot \bar{\boldsymbol{g}}_{\text{comp.}} = \boldsymbol{g}_{\text{DICs}} - \boldsymbol{S}_{\text{random}} \cdot \bar{\boldsymbol{g}}_{\text{random}}$
        Store $\bar{\boldsymbol{g}}_i$ in the population
    return Degenerate population $\{\bar{\boldsymbol{g}}_i\}_{i=1}^{M}$ with targeted spontaneous activity
```

Appendix D

Neuronal Activity Metrics and Comparison Methods

To describe neuronal activity more precisely, it is useful to introduce a set of quantitative metrics. In this work, these metrics serve both as descriptors and as core components of the training process for the deep learning pipeline through the auxiliary tasks. This appendix formalizes the definitions used. It also introduces the *van Rossum* distance that is used to compare quantitatively activity differences.

D.1 Neuronal Activity Metrics

In a computational setting, CBM simulations produce the full trace of neuronal activity, capturing the membrane potential evolution over time: $V(t)$. To simulate the constraint of real-world recordings that only capture spike times, we transform the voltage trace into a spike train:

$$V(t) \xrightarrow{\text{transformed into}} \boldsymbol{x} = [t_1, t_2, \ldots, t_{N_{\text{spikes}}}], \quad \text{where } t_1 < t_2 < \cdots < t_{N_{\text{spikes}}}.$$

Here, t_i represents the spike time of the i-th detected spike. Naturally, the total number of spikes, N_{spikes}, depends on the instance, its activity type, and the simulation (recording) duration.

Spike detection from the voltage trace is based on two threshold potentials: $V_{s1} = 10\,\text{mV}$ and $V_{s2} = 0\,\text{mV}$. A spike onset (t_b) is identified when the potential crosses above V_{s1}:

$$V(t_b) \leq V_{s1} < V(t_b + \delta t), \quad \forall \text{ sufficiently small } \delta t > 0.$$

Similarly, a spike ends at t_e when the potential falls below V_{s2}:

$$V(t_e) \geq V_{s2} > V(t_e + \delta t), \quad \forall \text{ sufficiently small } \delta t > 0.$$

The associated spike time is then defined as: $t_i = \frac{t_b + t_e}{2}$.

D.1.1 A measure for classification

Neuronal activity is often categorized as *spiking* or *bursting*. Previously in this work, classification was performed manually by inspecting traces and relying on intuitive definitions of *spikers* and *bursters*. Below, we propose a quantitative criterion for automatic classification of instances into these categories.

It is important to note that classification is not a strict requirement for the pipeline. In practice, this step helps guide the model during training, but its outcomes are not critical. Consequently, the decision threshold value is not extensively studied.

Classification is based on the *coefficient of variation* (CV) of the **interspike intervals** (ISIs). Given a sequence of spike times $\boldsymbol{x}$, the sequence of ISIs is computed as:

$$\boldsymbol{x} = [t_1, t_2, \ldots, t_{N_{\text{spikes}}}] \xrightarrow{\text{transformed into}} \boldsymbol{x}_{\text{ISI}} = [\Delta t_1, \Delta t_2, \ldots, \Delta t_{N_{\text{spikes}}-1}], \quad \text{where } \Delta t_i = t_{i+1} - t_i.$$

The coefficient of variation of the ISIs is then given by : $CV[\boldsymbol{x}_{\text{ISI}}] = \frac{\sigma[\boldsymbol{x}_{\text{ISI}}]}{\mu[\boldsymbol{x}_{\text{ISI}}]}$. Intuitively, a high coefficient of variation indicates a sequence where the intervals between spikes are highly variable (e.g., bursting, where long silent periods alternate with rapid spike bursts), whereas a low coefficient suggests relatively consistent spike intervals (e.g., tonic spiking). Classification follows the rule:

$$\texttt{type}[\boldsymbol{x}_{\mathrm{ISI}}] = \begin{cases} \text{silent}, & \text{if } \texttt{len}[\boldsymbol{x}_{\mathrm{ISI}}] \leq 2; \\ \text{spiking}, & \text{if } CV[\boldsymbol{x}_{\mathrm{ISI}}] \leq 0.1 \text{ and } \texttt{len}[\boldsymbol{x}_{\mathrm{ISI}}] > 2; \\ \text{bursting}, & \text{if } CV[\boldsymbol{x}_{\mathrm{ISI}}] > 0.1 \text{ and } \texttt{len}[\boldsymbol{x}_{\mathrm{ISI}}] > 2. \end{cases}$$

where `len` returns the sequence length. In practice, silent instances are removed from the dataset since we focus on spontaneously active neurons.

D.1.2 A spiking metric

To further compare spiking instances, we define the **spiking frequency**:

$$f_{\mathrm{spk}} = \frac{1}{\mu[\boldsymbol{x}_{\mathrm{ISI}}]} \quad (\mathrm{Hz}).$$

Our classification rule imposes an implicit lower bound on observable spiking frequency for a simulation duration $t \in [0, T]$. Specifically, $f_{\mathrm{spk}} > \frac{3}{T}$, otherwise, the activity is classified as silent.

D.1.3 Multiple bursting metrics

Bursting activity is more complex and involves more timescales. Therefore, several bursting descriptors are used jointly. These metrics rely on the ability to segment the spike train into successive bursts, meaning each spike is assigned to a particular burst. To achieve this, we use the fact that a burst is a sequence of spikes followed by a period of silence. We detect the first spike of each burst and then assign subsequent spikes to it until the next first spike of a burst is encountered. Practically, the first spike of a burst is identified as the spike for which the period since the last spike is too long to belong to the same burst. Since there are long interspike intervals (ISIs) during silent periods and short ISIs during firing, the distribution of ISIs is bimodal, and a criterion based on the midrange value of ISI can be applied:

$$\boldsymbol{x}_{\mathrm{ISI}} = [\Delta t_1, \Delta t_2, \ldots, \Delta t_{N_{\mathrm{spikes}}-1}] \xrightarrow{\text{transformed into}} \boldsymbol{x}_{\mathrm{burst}} = [\boldsymbol{B}_1, \boldsymbol{B}_2, \ldots, \boldsymbol{B}_{N_{\mathrm{burst}}}]$$
$$\boldsymbol{B}_i = [t_j, t_{j+1}, \ldots, t_k]$$
$$t_j \text{ is such that } \Delta t_{j-1} = t_j - t_{j-1} > \frac{\max[\boldsymbol{x}_{\mathrm{ISI}}] + \min[\boldsymbol{x}_{\mathrm{ISI}}]}{2}$$

Burst duration, frequencies and number of spikes per burst To characterize bursting activity further, we define several metrics:

- **Burst duration**: The duration of a burst is the time interval between the first and last spike of the burst. For a burst $\boldsymbol{B}_i = [t_j, t_{j+1}, \ldots, t_k]$, the duration is given by:

$$\mathrm{Duration}(\boldsymbol{B}_i) = t_k - t_j.$$

- **Intra-burst frequency**: This is the average firing rate within a burst, calculated as the number of spikes in the burst divided by the burst duration:

$$f_{\mathrm{intra}}(\boldsymbol{B}_i) = \frac{\texttt{len}(\boldsymbol{B}_i)}{\mathrm{Duration}(\boldsymbol{B}_i)}.$$

- **Inter-burst frequency**: This metric describes how often bursts occur by taking the inverse of the average time between the onsets of consecutive bursts:

$$f_{\mathrm{inter}} = \frac{1}{\frac{1}{N_{\mathrm{bursts}}-1} \sum_{i=1}^{N_{\mathrm{bursts}}-1} \left(t_{\mathrm{burst}_{i+1}} - t_{\mathrm{burst}_i}\right)}$$

- **Number of spikes per burst**: This metric simply counts the number of spikes within each burst, providing insight into the intensity of bursting activity :

$$\#\boldsymbol{B}_i = \texttt{len}(\boldsymbol{B}_i).$$

For the burst metrics, we need a careful handling of the simulation. Since we simulate for Tms, we may start in the middle of a burst or stop within a burst. Thus, to compute the bursting metrics, we only rely on the full bursts, removing the first and last burst of each sequence. The bursting activity is finally described by the average over the sequence of the different metrics.

D.2 Comparing Neuronal Activity Across Instances

To compare the neuronal activity of multiple instances, it is useful to employ a distance measure that captures the overall similarity or dissimilarity between spike trains without focusing on specific features. One such measure is the van Rossum distance (Houghton and Kreuz, 2012; Rossum, 2001).

Van Rossum distance The van Rossum distance is a metric designed to quantify the similarity between two spike trains. It transforms the spike trains into continuous functions using a kernel smoothing technique and then computes the Euclidean distance between these functions. The van Rossum distance between two spike trains $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as:

$$d_{\mathrm{VR}}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\int_0^\infty \left(f_{\boldsymbol{x}}(t) - f_{\boldsymbol{y}}(t)\right)^2 dt},$$

where $f_{\boldsymbol{x}}(t)$ and $f_{\boldsymbol{y}}(t)$ are the continuous functions obtained by convolving the spike trains $\boldsymbol{x}$ and $\boldsymbol{y}$ with a kernel function $K(t)$. A commonly used kernel is the causal exponential kernel:

$$K(t) = \mathbb{I}(t \geq 0)\frac{1}{\tau}e^{-t/\tau},$$

where $\mathbb{I}(\cdot)$ is the indicator function that returns 1 if the condition is true and 0 otherwise, and τ is a time constant that determines the width of the kernel. The choice of τ influences the sensitivity of the distance measure to the timing of spikes; a smaller τ makes the distance more sensitive to precise spike timings, while a larger τ smooths out the spike trains, focusing more on the overall firing rate.

The van Rossum distance enables the comparison of neuronal activity across instances without focusing on specific features such as spiking frequency or burst duration. Instead, it provides a holistic measure of similarity based on the entire spike train, making it a versatile tool for analyzing neuronal data.

This measurement will be particularly useful for assessing the overall performance of the pipeline and comparing the populations resulting from it with the input populations.

Making the distance insensitive to phase The van Rossum distance is not, by default, insensitive to the phase of spiking. This means that two identical activities starting at different absolute times are not considered equal by this metric. To address this issue, we propose performing a temporal alignment. This involves shifting the sequences to align the first spike of each before calculating the measure. Additionally, similar to the calculation of metrics, during bursting, we remove the first and last identified bursts in both sequences before aligning and comparing.

Appendix E

Pre-Generation Analyses and Rationale

This appendix provides detailed analyses and rationale for the choices made prior to the generation of degenerate populations of CBMs. It covers the selection of conductance distributions, threshold potential values, and the range of DIC values used in the study.

E.1 Conductance Distributions

This section describes the choices related to the distributions of maximal conductances, denoted as $\bar{\boldsymbol{g}} \sim \mathcal{D}$. In practice, we introduce two distinct distributions:

- $\mathcal{D}_{\text{analysis}}$: a distribution that covers a wide region of the conductances space, used for preliminary analyses prior to the generation of degenerate populations.

- $\mathcal{D}_{\text{generation}}$: a distribution whose density is concentrated on a smaller region of the conductances space, specifically employed during the generation of degenerate populations.

The use of these two different distributions is justified by their respective objectives. The distribution $\mathcal{D}_{\text{analysis}}$ is designed to explore a wide range of different instances to observe various properties of the CBM under study. It is advised to use a distribution that is broad enough to not lose information by limiting it to a smaller region of the space.

On the other hand, the distribution $\mathcal{D}_{\text{generation}}$ plays a crucial role in determining the *level of degeneracy*, meaning the size of the parameter space leading to degenerate behaviors. The goal is to have a distribution that is sufficiently spread out to observe a significant level of degeneracy, while still ensuring minimal negative impact on the *degenerate proximity*. Indeed, the variability that we observe in the firing pattern of populations generated by imposing the DIC values at threshold depends on the distribution used to sample the conductances that are not compensated. In other words, we want the instances obtained through the generation procedure based on the DICs theory to exhibit sufficiently similar activity to be categorized as degenerate ($\mathcal{D}_{\text{generation}}$ should be concentrated), while having a large variability in the conductance values ($\mathcal{D}_{\text{generation}}$ should be spread).

Figure 2.14 illustrates that some variability can exist within so-called '*degenerate*' populations, but this variability should remain sufficiently low. In other words, $\mathcal{D}_{\text{generation}}$ must spread its density as much as possible to facilitate degeneracy, while remaining concentrated enough to ensure that enforcing the DIC values at threshold enforces similar activities within the population.

A distribution for preliminary analyses To cover a large region of the conductances space, we follow the approach of Fyon et al., 2024 by using independent uniform distributions, with the bounds reported in Table E.2a. However, in contrast to the work of Fyon et al., the leak conductance is not sampled from a uniform distribution but instead from a Gamma distribution. This distribution is characterized by the following probability density function:

$$X \sim \text{Gamma}(k, \theta) \implies p_X(x) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}} \tag{E.1}$$

This choice is motivated by both the properties of the Gamma distribution and the impact of the leak conductance. First, the Gamma distribution is well-suited for modeling variables with strictly positive support, such as maximal conductances. Second, it does not impose an *a priori* upper bound on conductance values, which is considered beneficial given that the literature does not report a well-defined maximum limit.

Most importantly, with appropriate parameters, the gamma distribution assigns very little density near zero, unlike the uniform distribution. This property is particularly advantageous because it stabilizes normalization by the value of g_{leak}.

The shape parameter k and the scale parameter θ are chosen such that the first and second moments of the resulting Gamma distribution match those of the uniform distribution used in Fyon et al., 2024.

It is worth noting that, after conducting the analyses, additional tests using a uniform distribution produced very similar results, indicating that this choice is not critical.

A distribution for the generation procedure Following the approach of Fyon et al., a more concentrated distribution is used for the generation procedure. In addition to concentrating the probability mass, the generation distribution is designed to control the effect of *homogeneous scaling*, as introduced in Section 2.4.

In this approach, the conductances are no longer completely independent. The leak conductance is sampled first, and the other conductances are then scaled by the factor $\frac{g_{\text{leak}}}{g_{\text{leak,mean}}}$. The distributions used follow those of Fyon et al., 2024, meaning uniform distributions with bounds reported in Table E.2b.

Once again, we opt for a Gamma distribution for the leak conductance, applying the same fitting method based on the bounds from Fyon et al. Finally, no bounds are specified for the conductances $\bar{g}_{\text{Na}}, \bar{g}_{\text{Kd}}$, and $\bar{g}_{\text{H}}$ since these are not directly sampled but instead compensated during Step 1 of the generation procedure (Algorithm 4). In other words, we set $\bar{\boldsymbol{g}}_{\text{comp., spont.}} = (\bar{g}_{\text{Na}}, \bar{g}_{\text{Kd}}, \bar{g}_{\text{H}})$.

The DIC values associated with spontaneous activity Table E.1 reports the DIC values used, $\boldsymbol{g}_{\text{DICs, spont.}}$, during the first step of the generation procedure. These values are taken from Fyon et al., 2024. Additionally, it reports the $\boldsymbol{g}_{\text{DICs}}$ associated with the strong bursting, light bursting, and spiking populations from Figure 2.14. It also reports target DICs of Figure 2.16.

DIC values	g_{f}	g_{s}	g_{u}
$\boldsymbol{g}_{\text{DICs, spont.}}$	-6.2	4.0	5.0

(a) DIC values used for generating spontaneous activity. Those values are used in the first step of the generation procedure.

Target DIC values	g_{s}	g_{u}
$\boldsymbol{g}_{\text{DICs}}$ (*tonic spiking*)	4.0	5.0
$\boldsymbol{g}_{\text{DICs}}$ (*strong bursting*)	-4.0	5.0
$\boldsymbol{g}_{\text{DICs}}$ (*light bursting*)	-2.0	5.0
$\boldsymbol{g}_{\text{DICs}}$ (—)	-5.0	7.0

(b) Target DIC values for different neuronal activities. Those values are used to generate Figure 2.14 (first 3 rows) and Figure 2.16 (4th row).

Table E.1: DIC values associated with spontaneous neuronal activities. This table presents the DIC values used for generating spontaneous activity and the target DIC values for spiking, strong bursting, and light bursting activities.

Conductance $\bar{g}_i \sim \mathcal{U}(0; \bar{g}_{\text{max}})$	$\bar{g}_{\text{Na}}$	$\bar{g}_{\text{Kd}}$	$\bar{g}_{\text{CaT}}$	$\bar{g}_{\text{CaS}}$	$\bar{g}_{\text{KCa}}$	$\bar{g}_{\text{A}}$	$\bar{g}_{\text{H}}$
Maximum value $\bar{g}_{\text{max}}$	8000	350	12	50	250	600	0.7

Conductance $\bar{g}_i \sim \text{Gamma}(k, \theta)$	k	θ
g_{leak}	3	$\frac{1}{300}$

(a) Preliminary conductance distribution $\mathcal{D}_{\text{analysis}}$, used for broad exploratory analyses. Conductances are sampled independently from uniform distributions with predefined upper bounds, except for the leak conductance, which follows a gamma distribution.

Conductance $\bar{g}_i \sim \mathcal{U}(\bar{g}_{\text{min}}; \bar{g}_{\text{max}}) \frac{g_{\text{leak}}}{g_{\text{leak,mean}}}$	$\bar{g}_{\text{Na}}$	$\bar{g}_{\text{Kd}}$	$\bar{g}_{\text{CaT}}$	$\bar{g}_{\text{CaS}}$	$\bar{g}_{\text{KCa}}$	$\bar{g}_{\text{A}}$	$\bar{g}_{\text{H}}$
Minimum value $\bar{g}_{\text{min}}$	—	70	2	6	140	—	—
Maximum value $\bar{g}_{\text{max}}$	—	140	7	22	180	—	—

Conductance $\bar{g}_i \sim \text{Gamma}(k, \theta)$	k	θ
g_{leak}	27	$\frac{1}{2570}$

(b) Generation conductance distribution $\mathcal{D}_{\text{generation}}$, used to produce degenerate populations. This distribution is more concentrated and incorporates homogeneous scaling control by normalizing conductances relative to the leak conductance.

Table E.2: Distributions of maximal conductances in the STG model. Two distinct distributions $\bar{\boldsymbol{g}} \sim \mathcal{D}$ are used at different stages of the study: (a) $\mathcal{D}_{\text{analysis}}$, a broad distribution for preliminary analyses, and (b) $\mathcal{D}_{\text{generation}}$, a more concentrated distribution designed to control the level of degeneracy while maintaining degenerate proximity. Values are given such that the conductances are in $\text{mS}\,\text{cm}^{-2}$.

E.2 Threshold Potential and The DICs Space Range

E.2.1 Threshold Potential

An important value for this work and for the theory of DICs is the threshold potential, denoted as V_{th}. As introduced in Section 2.3, this value marks the membrane potential at which sensitivity to parameter variations most influences neuronal activity. Figure 2.10 shows that this threshold value, which can be identified as the first *decreasing* zero crossing of the total conductance $g_{\text{t}}(V)$, depends on the specific instance.

In this work, the theory of DICs is mainly used as an intermediary for the generation of degenerate populations. As a result, we do not have predefined instances and therefore lack prior knowledge of their precise threshold values. Our approach consists of identifying a default threshold value around which instances naturally distribute. This default value, denoted by V_{th} as a notation convenience, is used *a priori* as the threshold potential. The theory of DICs ensures that using an approximate rather than the '*exact*' (if that even makes sense) threshold value does not invalidate the method but only reduces its sensitivity. In other words, the DIC curves near the true threshold value are influenced by its local values, ensuring that the method remains applicable.

This section justifies the default threshold value of $V_{\text{th}} \approx -51\,\text{mV}$ used in this study. Notably, previous studies (Fyon et al., 2023; Fyon et al., 2024) have used a slightly different threshold value of $-50\,\text{mV}$. Our analysis confirms that $V_{\text{th}} \approx -51\,\text{mV}$ is a robust estimate, as it closely matches the mean and median of threshold potentials across sampled instances.

Specifically, we sample $M = 2000$ instances from the distribution $\mathcal{D}_{\text{analysis}}$ as an analysis dataset. For each instance, the total DIC curve $g_{\text{t}}(V)$ is computed following the formulation in Eq. (2.11) over a discrete set of potential values $V \in \{-100, -95, \dots, -5, 0\}$. These total conductance values are then used to define the initial bounds for a bisection procedure aimed at approximating the first decreasing root of $g_t(V)$ (Burden and Faires, 2010, Chapter 2). The bounds are identified as the first transition from $g_t > 0$ to $g_t < 0$. A maximum of 1000 iterations is allowed, with a stopping criterion requiring a tolerance of 10^{-6} on either V or g_t.

Figure E.1 presents the results of this procedure. The mean and median of the threshold potential are close to each other and approximately equal to $V_{th} \approx -51\,\text{mV}$, supporting our choice of default value. Furthermore, these values remain stable when the sample size is doubled ($M = 4000$), reinforcing confidence in the convergence of these results.

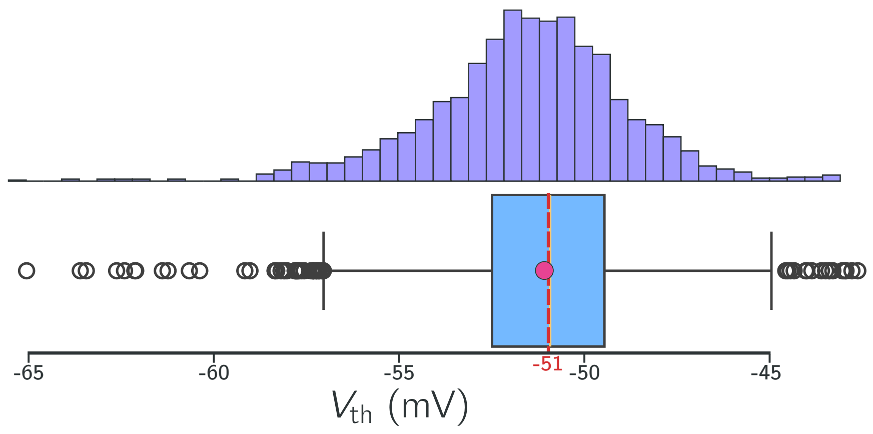


Figure E.1: Distribution of extracted threshold potentials in the STG model. The figure presents the distribution of threshold potentials V_{th} obtained from $M = 2000$ instances sampled from $\mathcal{D}_{\text{analysis}}$. The boxplot displays the variability of V_{th} across instances, with the median represented by the yellow line and the mean indicated by the pink circular marker. Outliers are shown as individual circles. The histogram above the boxplot provides a complementary view of the distribution, showing the density of extracted V_{th} values. The observed proximity between the mean and median supports the choice of the default threshold value $V_{th} \approx -51\,\text{mV}$, with the mean value being $-51.032\,\text{mV}$ and the median $-50.911\,\text{mV}$.

E.2.2 The DICs Space Range

The population generation procedure requires specifying the target DIC values *a priori*. This section identifies the range of these values to ensure a broad sampling of the DICs space when generating datasets for the deep learning pipeline. Once again, a numerical approach is used to establish empirical bounds.

The DIC values at the threshold potential are evaluated for the $M = 2000$ instances from the previous section. Figure E.2 illustrates the distributions of the slow and ultra-slow conductances obtained. Two main observations can be made from these results. First, the distribution of g_s differs significantly from that of g_u. While g_u is relatively uniformly distributed within the interval $[-2, 15]$, g_s follows a symmetric but non-uniform distribution centered around 0 within the interval $[-15, 15]$. Second, in the grid defined by $(g_s, g_u) \in [-15, 15] \times [-2, 15]$, the DIC space forms a single connected region, which arises naturally from the distribution of the conductances and the structure of the model.

Regarding negative ultra-slow DIC values Although the ultra-slow DIC distribution is uniform, we point out one of the constraints we had to apply in this work. In practice, negative values of g_u are observed from the distribution $\mathcal{D}_{\text{analysis}}$, but these values are not considered further in this study. Specifically, the range of g_u values is restricted to positive values. The reason for this choice stems from the observation that for negative values of g_u, the generated populations exhibit significant variability despite having the same DICs value at the threshold potential. We hypothesize that this ultra-slow positive feedback around threshold voltage completely destabilizes the neuron in all timescales around that voltage. Therefore, we choose not to consider this part of the DICs space.

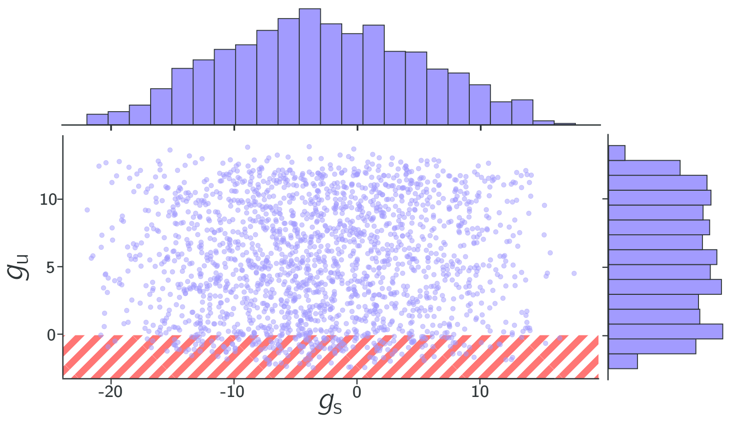


Figure E.2: Distribution of DIC values at the threshold potential in the STG model. This figure illustrates the distributions of the slow (g_s) and ultra-slow (g_u) conductances at the threshold potential, evaluated for $M = 2000$ instances sampled from the distribution $\mathcal{D}_{\text{analysis}}$. The scatter plot in the center shows the joint distribution of (g_s, g_u) pairs, with the red hatched region indicating the excluded range where $g_u < 0$. The histograms on the top and right margins display the marginal distributions of g_s and g_u, respectively. The slow conductance g_s follows a symmetric, non-uniform distribution centered around 0 within the interval $[-15, 15]$, while the ultra-slow conductance g_u is relatively uniformly distributed within the interval $[-2, 15]$. However, negative values of g_u are excluded in this study due to their association with increased variability in generated populations. The grid $(g_s, g_u) \in [-20, 20] \times [0, 20]$ is used for dataset generation to ensure broad coverage of the DICs space.

In addition, a closer analysis of the threshold potential reveals that the majority of outliers with a threshold potential that is significantly more polarized than $-51\,\text{mV}$ corresponds to negative values of g_u. Therefore, we hypothesize that for these instances, the default threshold potential should be shifted, as the current value is too far from the true threshold value.

A wider coverage for better generalization of results In the core part of the work, we choose to use the grid $(g_s, g_u) \in [-20, 20] \times [0, 20]$ for dataset generation. The goal is to ensure a sufficiently broad coverage of the interesting region in the DICs space.

Appendix F

The AdamW Optimizer and Training Details

This appendix provides a detailed overview of the AdamW optimizer and its configuration, as well as the implementation details of the training process for our models. We discuss the rationale behind our choice of optimizer, the specific configuration parameters used, and how these choices contributed to the efficient and effective training of our models. Additionally, we outline the implementation details, including the use of the PyTorch framework and the computational resources leveraged during training. Finally we detail the computation of importance and correlations values reported in the hyperparameters tuning section.

F.1 Configuration of the AdamW Optimizer

In this work, we employed the AdamW optimizer to train our models (Loshchilov and Hutter, 2019). AdamW is an enhanced variant of the Adam optimizer, designed to better handle weight regularization, which is crucial for preventing overfitting. The configuration of AdamW and the associated learning rate scheduler is as follows:

The optimizer is set up with a learning rate η, which controls the size of the weight updates at each iteration. The parameters β_1 and β_2 control the decay rates for the first and second moment estimates used in the calculations of the optimizer. Here, we use $\beta_1 = 0.9$ and $\beta_2 = 0.98$. The value of β_1 is the default value, for β_2 we explicitly put it to a smaller value (default is 0.999) because it helps to have a much more stable training. The parameter for weight decay is used for L2 regularization, helping to prevent overfitting by penalizing large weight values. We use $\lambda_{\text{weight decay}} = 0.04$ which is a bit bigger than the default value of 0.01.

For the learning rate scheduler, we used a cosine annealing schedule with warm restarts, which adjusts the learning rate according to a cosine function with periodic restarts (Fig. F.1). The learning rate η_t at epoch t is given by:

$$\eta_t = \eta_{\text{min}} + \frac{1}{2}\left(\eta - \eta_{\text{min}}\right)\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_i}\pi\right)\right)$$

Where:

- η is the initial learning rate. This value is tuned through the hyperparameters tuning.
- η_{min} is the minimum learning rate. We use $\eta_{\text{min}} = \frac{\eta}{10}$.
- T_{cur} is the number of epochs since the last restart.
- T_i is the number of epochs between two restarts. We use $T_i = 10$ epochs.

Advantages of the AdamW Optimizer AdamW incorporates several mechanisms to enhance convergence efficiency and reduce the risk of overfitting:

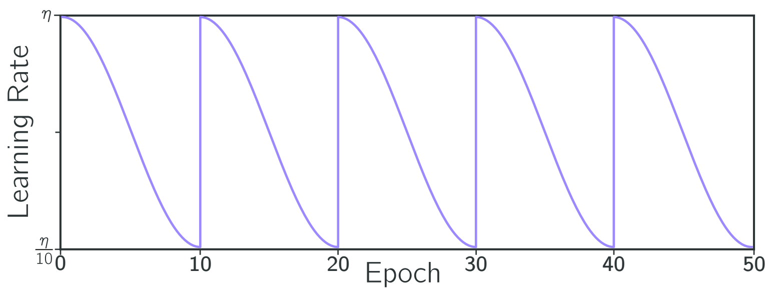


Figure F.1: Cosine Annealing Learning Rate Schedule with Warm Restarts. This figure illustrates the learning rate schedule used during training, which follows a cosine annealing pattern with periodic restarts. The learning rate decreases according to a cosine function and resets at regular intervals, allowing for better convergence and stability during training.

- **Momentum**: AdamW keeps track of the average direction of past gradients — not just the current one — which helps the optimizer build up speed in useful directions and dampen oscillations. This is achieved through an exponentially decaying average of past gradients, allowing it to "remember" past updates and move more smoothly across the loss landscape.

- **Adaptive learning rates**: Instead of using the same learning rate for all parameters, AdamW adjusts the learning rate for each one based on how large or small its past gradients have been. Parameters with consistently large gradients get smaller steps, while those with small gradients can take larger steps. This is done by maintaining a running average of the squared gradients norms, allowing the optimizer to scale updates dynamically and more effectively handle varying curvature in the loss surface.

- **Regularization**: AdamW improves regularization by applying weight decay directly to the parameters after the gradient update, rather than adding it to the gradients. This "decoupled" approach ensures that weight decay acts as a true penalty on parameter norm, helping to keep weights small, reduce model complexity, and improve generalization — all without interfering with the adaptive update mechanism.

F.2 Implementation Details

The implementation of our models was carried out using the PyTorch framework, which provides a flexible and efficient platform for developing deep learning models (Paszke et al., 2017). While PyTorch offers a wide range of pre-built components, we chose to implement most of the building blocks from scratch to ensure a deep understanding and customization of the model architecture.

Custom implementations Most components of our model, including the neural network layers and loss calculations, were implemented from scratch. This approach allowed us to tailor the architecture to our specific needs and gain insights into the behavior of each component. By writing our own implementations, we were able to experiment with different designs and optimizations, ultimately leading to a more efficient and effective model. Additionally, this choice was driven by the belief that a Master Thesis is primarily an opportunity to acquire new knowledge and deepen understanding. The only major exception is the Multi-Head Attention layer, which was used directly from the built-in implementation.

Training on the Alan cluster Our training process was significantly enhanced by utilizing the Alan GPU Cluster at the University of Liège. Access to the cluster GPU resources allowed us to accelerate the training process. To give the reader an idea, the longest training session took just less than 24 hours. We argue that this time is mainly due to the size of the STG dataset and the very small batch size used. The model itself is relatively small.

F.3 Computation of Hyperparameter Importance and Correlations

In Section 6.1, we present the importance and correlation values of the hyperparameters studied. These values are computed using different methods tailored to their respective purposes.

Importance of hyperparameters The importance of each hyperparameter is determined by fitting a random forest model to the hyperparameter tuning data. Random forests are ensembles of decision trees, where each tree is trained on a random subset of the data and considers a random subset of features (hyperparameters) at each split. The importance of a hyperparameter is calculated based on how often it is used to make key decisions across all the trees in the forest (Louppe et al., 2013).

In other words, importances are computed by solving a new supervised learning problem, where the inputs are the hyperparameter values and the output is the model best validation loss. This approach is powerful because random forests can fit linear and non-linear relationships and do not assume *a priori* a specific functional form between the input and the output.

Correlation of hyperparameters In addition to importance, we also analyze the correlation of hyperparameters. Correlation refers to the (Pearson's) linear correlation between the hyperparameter value and the best validation loss. A high correlation means that when the hyperparameter has a higher value, the metric also tends to have higher values, and vice versa. This provides an overview, on average, of how changes in hyperparameter values affect the validation loss. Correlation is a valuable metric to consider because it can help make quick choices about which hyperparameters to adjust, especially when time and computational resources are limited. However, it cannot capture second-order interactions between inputs. Therefore, while correlation offers useful insights and aids in rapid decision-making, it should be interpreted with caution and in the context of other metrics and analyses (such as the full random search that we are doing in this work).

We do not provide a correlation for the activation function because it is a categorical variable, and linear correlation measures are not applicable to non-ordered categorical data. For the log transformation, '`true`' is considered as 1 and '`false`' as 0.

Appendix G

Transfer Details

This appendix briefly summarizes the details related to the transfer of the method from the STG neuron model to the DA neuron model.

G.1 Preliminaries

By applying the same procedure as for the STG model, we identified that for the DA model, the threshold potential value is $V_{\text{th}} \approx -55.5\,\text{mV}$. The range in the DICs space is $(g_s, g_u) \in [-10; 15] \times [0; 20]$. The distributions used, as well as the DIC values associated with spontaneous activities, are those from Fyon et al., 2024. A Gamma distribution is used for the leak conductance, and uniform distributions are used for the other conductances. Since there is no explicit modeling of ionic dynamics, the iterative procedure reduces to the original procedure by Fyon et al. The reachability analysis allows us to identify the following heuristic:

$$\bar{\mathbf{g}}_{\text{comp.}} = \begin{cases} (\bar{g}_{\text{ERG}}, \bar{g}_{\text{CaL}}), & \text{if } g_s < 0 \\ (\bar{g}_{\text{ERG}}, \bar{g}_{\text{Kd}}), & \text{if } g_s > 0 \end{cases} \tag{G.1}$$

Note that we sometimes observe reachability issues for $g_u < 1.5$.

G.2 Modification of the Architecture and Training

G.2.1 The Architecture

To perform the transfer, we introduce LoRA adapters in different layers of the architecture. In practice, satisfactory performance is achieved by focusing solely on the linear layers. In other words, we do not introduce LoRA in the attention layers. Figure G.1 illustrates the adapted pipeline. The majority of the parameters are frozen (also see Fig. 3.2), and the weights are derived from the training on the STG model. The transfer introduces a new hyperparameter, r, which represents the dimension of the adapter. This is the only hyperparameter that we retune via a grid search. We train the pipeline for the following values of r and evaluate on the validation set: $r \in \{2, 4, 8, 16, 32, 48, 64\}$. See Section 6.3 for the results.

G.2.2 Training

A preliminary exploration of the dataset generated following the same procedure as for the STG model highlights that the dataset for the DA model is much less balanced. Within the proposed range, we find 58% of silent instances (which are excluded), 34% of instances in spiking (fast spiking or slow pacemaking), and only 8% in bursting.

We then use a balanced cross-entropy loss for the classification head, and the reported accuracy is a balanced accuracy. During dataset generation, the number of populations generated is chosen such that approximately 25, 000 populations are active in practice after discarding silent instances.

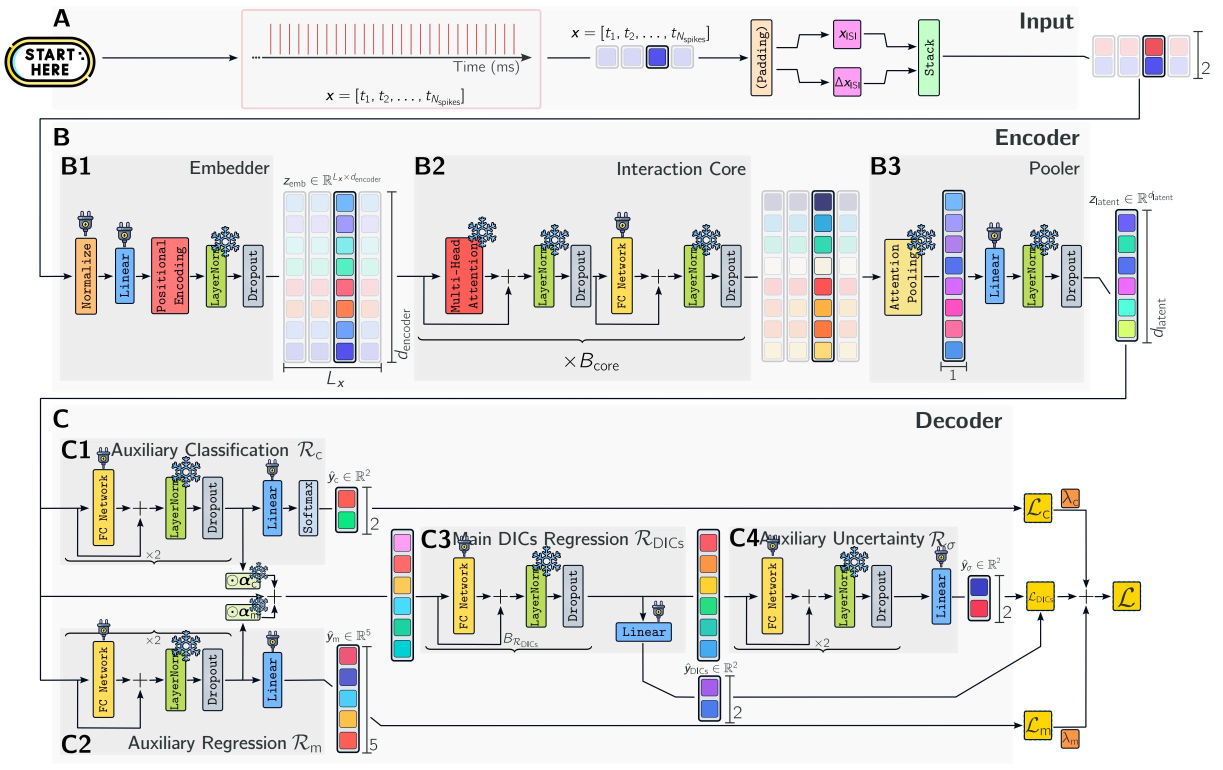

A
START HERE
Time (ms)
$x = [t_1, t_2, \dots, t_{N_{\text{spikes}}}]$
(Padding)
x_{ISI}
Δx_{ISI}
Stack
Input
2
B
Encoder
B1 Embedder
Normalize
Linear
Positional Encoding
LayerNorm
Dropout
$z_{\text{emb}} \in \mathbb{R}^{L_x \times d_{\text{encoder}}}$
d_{encoder}
L_x
B2 Interaction Core
Multi-Head Attention
FC Network
$\times B_{\text{core}}$
B3 Pooler
Attention Pooling
$z_{\text{latent}} \in \mathbb{R}^{d_{\text{latent}}}$
d_{latent}
C
Decoder
C1 Auxiliary Classification $\mathcal{R}_{\text{c}}$
Softmax
$\hat{y}_{\text{c}} \in \mathbb{R}^2$
×2
$\odot\alpha_{\text{c}}$
$\odot\alpha_{\text{m}}$
C2 Auxiliary Regression $\mathcal{R}_{\text{m}}$
$\hat{y}_{\text{m}} \in \mathbb{R}^5$
C3 Main DICs Regression $\mathcal{R}_{\text{DICs}}$
$B_{\mathcal{R}_{\text{DICs}}}$
$\hat{y}_{\text{DICs}} \in \mathbb{R}^2$
C4 Auxiliary Uncertainty $\mathcal{R}_{\sigma}$
$\hat{y}_{\sigma} \in \mathbb{R}^2$
$\mathcal{L}_{\text{c}}$
λ_{c}
$\mathcal{L}_{\text{DICs}}$
$\mathcal{L}$
$\mathcal{L}_{\text{m}}$
λ_{m}

Figure G.1: Adapted Architecture with LoRA Adapters. Illustration of the pipeline adapted with LoRA adapters for transferring the model from the STG neuron to the DA neuron. The plug icon indicates the insertion of a LoRA adapter, where the original layer parameters are frozen and a new linear projection is added in an additive manner with new learnable parameters (Fig. 3.2). The freeze icon signifies that the parameters of the corresponding layer are not retrained, preserving its original functionality and weights. The only exception is the normalization block (orange) at pipeline input. This one is not adapted by a LoRA, but directly replaced by the new normalization statistics calculated on the DA train set.