Machine Learning for Inference in Biophysical Neuroscience Simulations

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Michael Deistler
aus Landau a. d. Isar

Tübingen
2025

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	30.07.2025
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Jakob H. Macke
2. Berichterstatter:	Prof. Dr. Philipp Hennig
3. Berichterstatter:	Prof. Dr. Dan Goodman

ABSTRACT

A central challenge in neuroscience is that many properties of neural systems cannot be measured exactly. This limits our understanding of these systems and our ability to build simulations that match experimental recordings or predict neural responses to unseen stimuli. *Inference* allows scientists to identify parameters—the properties that cannot be measured exactly—such that biophysical simulations are consistent with experimental measurements of neural activity. However, previous inference methods struggle with complex biophysical neuroscience simulations or can infer only a limited number of parameters.

This thesis presents new inference methods for biophysical simulations in neuroscience. To overcome limitations of previous methods, we leverage recent advances in machine learning, particularly in neural density estimation and automatic differentiation. By inferring biophysical properties, we open up possibilities to build accurate biophysical simulations of neural activity, to study parameter degeneracy, and to gain insight into properties of neural systems. While this thesis focuses on biophysical neuroscience simulations, many of the developed methods are broadly applicable and are already being used across various scientific disciplines.

The thesis consists of two main parts. In the first part, we develop and apply methods for neural simulation-based Bayesian inference (SBI). SBI uses neural networks to invert mechanistic computer simulations, thereby providing Bayesian estimates of parameters given experimental measurements. We use these methods to study how a natural constraint on neural circuits—low metabolic cost—constrains circuit properties. We then develop new methods for improving the flexibility, robustness and efficiency of SBI. Finally, we present `sbi`, a Python toolbox for simulation-based Bayesian inference which implements many popular SBI methods and allows domain scientists to apply these methods to their simulators and measurements.

In the second part, we show that differentiable simulation enables the identification of parameters that align biophysical simulations with experimental recordings or computational tasks. We develop Jaxley, the first differentiable biophysics simulator for neuroscience and use it to fit parameters of biophysical simulations with gradient descent. This approach scales parameter inference to large-scale biophysical models, including morphologically detailed single-cell and network models, and demonstrates that differentiable simulation can overcome previous limits on the number of parameters.

Overall, the presented methods and results demonstrate that machine learning unlocks new possibilities for constructing biophysical simulations in neuroscience. By overcoming a central challenge—inferring parameters from measurements of neural activity—we hope that our methods will enable new insights into the cellular and synaptic contributions to biological intelligence.

ZUSAMMENFASSUNG

Eine zentrale Herausforderung in den Neurowissenschaften besteht darin, dass viele Eigenschaften neuronaler Systeme nicht genau gemessen werden können. Dies schränkt unser Verständnis dieser Systeme und unsere Fähigkeit ein, Simulationen zu erstellen, die mit experimentellen Messungen übereinstimmen oder neuronale Reaktionen auf neue Stimuli vorhersagen. Durch Inferenz können Parameter—die nicht exakt messbaren Eigenschaften—identifiziert werden, so dass biophysikalische Simulationen mit experimentellen Messungen der neuronalen Aktivität übereinstimmen. Bisherige Inferenzmethoden haben jedoch Schwierigkeiten mit komplexen biophysikalischen Simulationen oder können nur wenige Parameter schätzen.

Diese Arbeit präsentiert neue Inferenzmethoden für biophysikalische Simulationen in den Neurowissenschaften. Um die Einschränkungen bisheriger Methoden zu überwinden, nutzen wir Fortschritte im Bereich des maschinellen Lernens, insbesondere bei der generativen künstlichen Intelligenz und bei der automatischen Differenzierung. Indem wir biophysikalische Eigenschaften so ableiten, dass sie mit experimentellen Aufzeichnungen übereinstimmen, eröffnen wir die Möglichkeit, präzise biophysikalische Modelle der neuronalen Aktivität zu erstellen, die Degeneration von Parametern in biophysikalischen Modellen zu untersuchen und Eigenschaften neuronaler Systeme abzuleiten, die noch nicht gemessen werden können. Obwohl sich diese Arbeit auf biophysikalische neurowissenschaftliche Simulationen konzentriert, sind viele der entwickelten Methoden auf ein breites Spektrum wissenschaftlicher Disziplinen anwendbar—und werden dort aktiv eingesetzt.

Die Arbeit besteht aus zwei Hauptteilen. Im ersten Teil werden Methoden zur neuronalen simulationsbasierten Bayes'schen Inferenz (SBI) entwickelt und angewendet. SBI verwendet neuronale Netze, um mechanistische Computersimulationen zu invertieren und dadurch Bayes'sche Schätzungen von Parametern gegeben experimentellen Messungen zu liefern. Wir verwenden diese Methoden, um zu untersuchen, wie eine natürliche Einschränkung neuronaler Schaltkreise—niedrige metabolische Kosten—die Eigenschaften der Schaltkreise einschränkt. Anschließend entwickeln wir neue Methoden zur Verbesserung der Flexibilität, Robustheit und Effizienz von SBI. Schließlich stellen wir `sbi` vor, eine Python-Toolbox für simulationsbasierte Bayes'sche Inferenz, die viele gängige SBI-Methoden implementiert und es Fachwissenschaftlern ermöglicht, diese Methoden auf ihre Simulatoren und Messungen anzuwenden.

Im zweiten Teil zeigen wir, dass differenzierbare Simulationen neue Möglichkeiten zur Identifizierung von Parametern biophysikalischer Modelle eröffnen. Wir entwickeln den ersten differenzierbaren Biophysik-Simulator für die Neurowissenschaften und verwenden ihn, um Parameter mit Gradientenabstieg zu trainieren. Dies ermöglicht es uns, biophysikalische Modelle so anzupassen, dass sie mit experimentellen

Aufzeichnungen übereinstimmen oder Aufgaben lösen, wobei die Anzahl der Parameter praktisch unbegrenzt ist. Der differenzierbare Biophysik-Simulator, JAXLEY, wird es experimentellen und rechnergestützten Neurowissenschaftlern ermöglichen, diese Methoden auf ihre Simulationen und Aufgaben anzuwenden.

Insgesamt zeigen die vorgestellten Methoden und Ergebnisse, dass maschinelles Lernen neue Möglichkeiten für die Erstellung biophysikalischer Modelle in den Neurowissenschaften eröffnet. Wir hoffen, dass unsere Methoden neue Einblicke in die zellulären und synaptischen Beiträge zur biologischen Intelligenz ermöglichen, indem sie eine zentrale Herausforderung dieser Modelle überwinden—die Identifizierung von Parametern, die der neuronalen Aktivität entsprechen.

ACKNOWLEDGEMENTS

It has been almost five years since I started my PhD. I am already whimsical looking back at those years, and I believe that I will always remember these years with joy. By and large, this will be because of the fantastic people I met along the way, and I want to express my deepest gratitude to everybody who supported me and made this a memorable time.

First and foremost, I want to thank my advisor, Jakob H. Macke. Thank you for your guidance, for your trust, and for inspiring me. Thank you for the freedom you have given me, and for having allowed me to travel and become part of a community. Finally, thank you for building an environment that is not only a place to work but a fun place to be in.

I want to thank Pedro J. Gonçalves for six years of weekly meetings with advice, support, and good laughs. Thank you for being the best "post-doc" supervisor I could have wished for.

I want to extend this gratitude to all lab members, particularly to Richard Gao, Auguste Schulz, Janne K. Lappalainen, and Poornima Ramesh—you are a fantastic crowd and made this journey so much more enjoyable.

I want to thank Philipp Hennig, for his support and feedback in TAC meetings and over coffee breaks, and for serving as evaluator of my thesis. I want to thank Philipp Berens for inspiring scientific discussions and for guidance on how to grow up in science. I want to thank Franziska Weiler for her administrative support.

I would like to thank my early mentors in the lab: Jan-Matthis Lueckmann, Álvaro Tejero-Cantero, and David S. Greenberg, for teaching me how to make figures, write papers, and develop software. I want to thank many amazing people with whom I worked closely—almost on a day-to-day basis—over many years: Jan Boelts, Manuel Gloeckler, Richard Gao, Jonas Beck, and Kyra L. Kadhim, thank you for inspiring discussions and for the joy of working together. Next, I want to thank my co-authors and collaborators throughout this thesis: Ziwei Huang, Matthijs Pals, Cornelius Schröder, Janne K. Lappalainen, Yves Bernaerts, Guy Moss, and many others. I also want to thank my fantastic M.Sc. students Manuel Gloeckler, Jonas Beck, Mila Gorecki, Florian Schönleitner, Bálint Mucsányi, and Dorá Molnár. I would like to thank Ehsan Kazemi, who supervised me during my internship, for his support, hospitality, and enthusiasm. I would also like to thank the entire community that has evolved around the `sbi` toolbox: I am truly proud to be part of this community.

Finally, with all my heart, I want to thank my friends and family, especially my parents and grandparents, my sister, and Valeriia.

AUTHOR CONTRIBUTIONS

All text in the main part of this thesis was written by myself. All figures in Chapters 1 and 2 were drawn by myself and do not appear elsewhere. The figures in Chapters 3 and 4 are adapted from publications included with this thesis. All of these figures were plotted or drawn by myself.

Some results presented in this thesis were created in joint work between me and co-authors of the publications that are included with this thesis. For detailed author contributions, see Appendix B.

CONTENTS

1

INTRODUCTION

Even with the most advanced experimental techniques, many quantities in nature cannot be measured accurately. What is the mass of a black hole? How many ion channels are contained in the tiniest structures of neurons? The lack of knowledge of these properties limits our understanding of nature, and it prevents scientists from building computer simulations which accurately mimic natural processes.

In many cases, however, it is possible to measure phenomena, signals, or quantities that are related to these properties. For example, the mass of a black hole influences its gravitational wave, and ion channels impact the voltage response of a neuron. *Inference* is the process of estimating parameters—the properties that cannot be measured—given such measurements[1] (Fig. 1.1).

Any inference is based on a theory (or *model*) of how parameters impact measurements. In science, these models can be incredibly complex. For example, in neuroscience, a model of the voltage response of a neuron given the exact properties of a cell involves interactions of many proteins and ions, each with their own dynamics. As models become increasingly complicated, scientists use computer *simulations* to implement them. Computer simulations allow scientists to evaluate complex theories, but they can be slow to simulate, highly non-linear, and stochastic, all of which make inference for computer simulations in science challenging.

In this thesis, we develop methods that enable inference for complex and stochastic computer simulations. The resulting parameter estimates can be used to compare system properties underlying different experimental conditions or to detect abnormalities in data. In addition, inference in computer simulations identifies simulations that are consistent with observations. Such simulations can provide an understanding of the mechanisms underlying measurements and can predict responses to unseen inputs, conditions, or stimuli, thereby guiding experimental protocols.

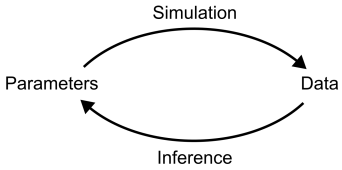


FIGURE 1.1: **Parameter inference.** Simulation generates predictions given a set of properties that cannot be measured exactly (parameters). *Inference* estimates parameters given data.

1 In the following, we use the term *parameter* to emphasize the target of inference and we use *property* to remind the reader that parameters of scientific models correspond to interpretable real-world quantities.

1.1 BIOPHYSICAL SIMULATIONS IN NEUROSCIENCE

In neuroscience, biophysical computer simulations have long played a central role in studying the cellular and synaptic mechanisms underlying neural activity. Perhaps the first biophysical model dates back to Alan Hodgkin and Andrew Huxley who, in a series of five seminal papers published in the Journal of Physiology in 1952, developed a quantitative model of the action potential [1, 6–9]. Hodgkin and Huxley recorded ionic currents from the giant axon of a squid and described the voltage dependence of two of these ionic currents, sodium and potassium. Building on this, they developed a biophysical model of action potential generation [1]. Remarkably, due to their measurements of the voltage dependence of the ion channels, the model contained only two free parameters: Merely the density of sodium and potassium conductances could not be measured and had to be tuned such that the biophysical simulation would match voltage recordings. Hodgkin and Huxley fitted those values manually, by hand [10].

The Hodgkin–Huxley model became the starting point for decades of biophysical modeling of neural systems, with models ever growing in the amount of detail, the variety of ion channels, and the number of neurons (Fig. 1.2). Ten years after Hodgkin and Huxley, in 1962, Rall [2] developed the first computational model which described the voltage of a neuron not just at one particular location, but described how the voltage evolves throughout the entire branched morphology of a neuron. In the years following the model of Rall, many toolboxes were developed that built on his approach, allowing increasingly large-scale simulations of biophysical models. Some of those toolboxes, although by now more than three decades old, are still being actively maintained: For example, the NEURON toolbox

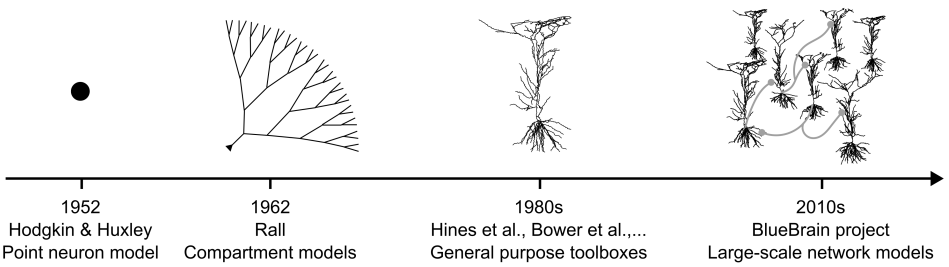


FIGURE 1.2: **The evolution of biophysical simulation.** In 1952, Hodgkin & Huxley [1] developed a biophysical model of action potential generation in the giant axon of the squid. Their simulations were based on a point neuron model. In 1962, Rall [2] developed the first branched compartment model to study the impact of dendrites on neuronal function. In the 1980s, multiple general software toolboxes were developed that allowed a more wide-spread adoption of morphologically detailed neuron modeling. Prominent examples are NEURON [3], GENESIS [4], or NODUS [5]. In the 2010s, the BlueBrain project, a team of hundreds of scientists and engineers, built a large-scale biophysical model of a cortical column containing tens of thousands of morphologically detailed neurons.

developed by Hines & Carnevale [3], which implements numerical routines [11] that can accurately and efficiently simulate morphologically detailed biophysical models.

Nowadays, more than half a decade after the first simulations of Hodgkin and Huxley, teams of hundreds of researchers are building biophysical networks consisting of tens of thousands of biophysical neuron models, each of which simulates voltage activity of every branch of every neuron and includes a wide variety of ion channels [12]. The scale and detail make these models increasingly flexible and enable scientists to simulate network phenomena with biophysical detail, but they also create a challenge: As biophysical models grow in size and detail, they contain dozens, hundreds, or even many thousands of parameters. In order to gain insights into these properties and to build faithful biophysical models, we require methods that infer these parameters from measurements of neural activity.

1.2 PARAMETER INFERENCE IN BIOPHYSICS

How can we perform inference in large-scale biophysical models? Inferring dozens, hundreds, or thousands of parameters of biophysical models—and scientific computer simulations in general—is challenging. A fundamental difficulty for parameter inference is the curse of dimensionality: The volume of the parameter search space increases exponentially with the number of parameters. In addition, unlike statistical models such as deep neural networks, parameter inference is particularly difficult for scientific simulators, for a number of reasons. First, these simulators can be slow to simulate. For example, large-scale biophysical systems can take many seconds or minutes to run just once, and fitting approaches typically require many simulation runs. In particular, some fitting approaches such as Markov-chain Monte–Carlo require many *sequential* runs of the simulator, leading to very slow fitting. Second, scientific simulators can lead to highly irregular loss surfaces. Even small changes in parameters can lead to a large difference in the simulation result. This makes it difficult to study global trends of how changes in parameters lead to changes in the simulation result. In addition, it can lead to many local optima, which "trap" fitting methods and prevent them from finding global optima. Third, depending on the noise model, the likelihood of data given parameters, a crucial ingredient to many approaches to inference, cannot be evaluated. For those models, inference must be able to deal with stochastic and noisy simulations only. Fourth, for many scientific simulators, one cannot easily compute the gradient with respect to parameters. For example, prior to this thesis, no toolbox for biophysical simulation existed which could evaluate the gradient. This prevents gradient-based fitting methods such as gradient descent or Hamiltonian Monte–Carlo. Finally, for many scientific simulators, many parameter sets are able to fit the data well [13]. This presents a significant challenge for fitting methods because a single "best-fitting" parameter set might not generalize well to new or unseen inputs.

Together with the curse of dimensionality, these issues make it difficult to adjust parameters of scientific simulators, posing challenges to domains ranging from

astrophysics, particle physics, geoscience and evolutionary biology to epidemiology and neuroscience.

As such, prior to this thesis, inference methods for biophysical neuron simulations either dealt only with some subsets of noise models (such that the likelihood becomes available) or with simulations of moderate size (such that the number of parameters remains small and can also be fit by gradient-free inference methods), or they returned only a few well-fitting parameter sets (as compared to the full space of solutions). The starting point for this thesis was to improve parameter inference methods for biophysical simulations in particular, but with applications to nearly all domains in the quantitative sciences. To achieve this, we capitalized on recent advances in machine learning.

1.3 MACHINE LEARNING FOR SIMULATION-BASED PARAMETER INFERENCE

Over the past years, machine learning, and in particular deep learning, has revolutionized many areas of our daily lives. The first successes of deep learning were in the field of computer vision [14] but, with time, scientists also started to exploit the potential of machine learning for scientific discovery [15].

In simulation-based Bayesian inference, recent methods use conditional deep neural density estimators to predict data-matching parameters [16]. This field was relatively small at the time of starting this thesis (28 papers in 2019), but it has continually grown into a larger research community (193 papers in 2024) [17]. The developed tools are used by domain scientists across a range of domains from particle physics [18], biology [19], neuroscience [20–22], and astrophysics [23–25].

In addition, progress in machine learning, and in particular in deep learning, is fueled by programming languages that support *automatic differentiation* [26]. These programming languages can compute, without manual labor from the user, the gradient through arbitrary computer programs. In deep learning, this has allowed researchers to build and explore flexible and powerful neural network architectures, but it also bears large potential outside of deep learning: By re-implementing existing simulators in differentiable programming languages, one can compute the gradient of simulations with respect to their parameters. This idea has empowered simulators in many domains of science [27–31].

Both of these methods have led to many exciting opportunities for building a new generation of scientific models. A recent review titled "Simulation intelligence: Towards a new generation of scientific methods" [15] predicts:

> The continued blending of data-driven, AI, simulation, and large-scale computing workflows is a promising path towards major progress and the emergence of a new scientific methodology.

This thesis contributes to this endeavor by developing machine learning methods for identifying parameters of simulation-based models, in particular for biophysical models of neural dynamics. This thesis is made up of six publications. These publications describe methods, applications, and software for simulation-based

Bayesian inference, as well as the first differentiable simulator for biophysical neuron modeling and the demonstration that it can be used to fit these models with gradient descent. Overall, this thesis demonstrates possibilities opened up by machine learning for building more accurate, faithful, and large-scale biophysical simulations in neuroscience.

1.4 OUTLINE OF THIS THESIS

In the introduction, we outlined the motivation for the methods that are developed in this thesis. Chapter 2 provides necessary background. The main part of the thesis is then made up of two chapters. In Chapter 3, we describe new methods and applications for simulation-based Bayesian inference. First, using these methods, we will describe our finding that disparate parameter sets, each of which has the same network activity, can have vastly different metabolic costs. Second, we will describe a new algorithm which makes active learning for neural simulation-based Bayesian inference more robust and applicable. Third, we will present a new method that aims to make scientific discovery with Bayesian inference more robust by relying on generalized Bayesian inference. Fourth, we will present a software toolkit which enables domain scientists to use modern simulation-based Bayesian inference methods. In Chapter 4, we will describe the first differentiable simulator for biophysical neuron models. Using this simulator, we will demonstrate that gradient descent enables training large-scale biophysical models and overcomes previous limitations on the number of parameters. Finally, in Chapter 5, we will discuss possibilities opened up by this thesis and provide an outlook and conclusions. The Appendix includes all publications and outlines author contributions to each of these publications. Below is a list of the six publications included with this thesis, as well as an additional set of 14 publications to which I contributed throughout my PhD, but which are not included in this thesis.

List of publications included in this thesis

The following publications are included in this thesis:

Michael Deistler, Jakob H. Macke[‡], Pedro J. Gonçalves[‡] (2022), Energy efficient network activity from disparate circuit parameters, *PNAS* [32]

Michael Deistler, Pedro J Gonçalves[‡], Jakob H Macke[‡] (2022), Truncated proposals for scalable and hassle-free simulation-based inference, *NeurIPS* [33]

Richard Gao[†], **Michael Deistler**[†], Jakob H Macke (2023), Generalized Bayesian Inference for Scientific Simulators via Amortized Cost Estimation, *NeurIPS* [34]

Álvaro Tejero-Cantero[†], Jan F Boelts[†], **Michael Deistler**[†], Jan-Matthis Lueckmann[†], Conor Durkan[†], Pedro J Gonçalves, David S Greenberg, Jakob H Macke (2020), `sbi`: A toolbox for simulation-based inference, *Journal of Open Source Software* [35]

Jan Boelts†, **Michael Deistler**†, ..., Jakob H Macke (2024), `sbi` reloaded: A toolkit for simulation-based inference workflows, *arxiv* [36]

Michael Deistler, Kyra L Kadhim, Jonas Beck, Matthijs Pals, Ziwei Huang, Manuel Gloeckler, Janne K Lappalainen, Cornelius Schröder, Philipp Berens, Pedro J Gonçalves, Jakob H Macke (2024), Differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics, *bioRxiv* [37]

List of publications not included in this thesis

I contributed to the following publications throughout my PhD, but they are not included in this thesis:

Richard Gao, **Michael Deistler**, Auguste Schulz, Pedro J Gonçalves, Jakob H Macke (2024), Deep inverse modeling reveals dynamic-dependent invariances in neural circuit mechanisms, *bioRxiv* [38]

JP Manzano-Patrón, **Michael Deistler**, Cornelius Schröder, Theodore Kypraios, Pedro J Gonçalves, Jakob H Macke, Stamatios SN Sotiropoulos (2024), Uncertainty mapping and probabilistic tractography using Simulation-Based Inference in diffusion MRI: A comparison with classical Bayes, *bioRxiv* [22]

Hamid Motallebzadeh, **Michael Deistler**, Florian M Schönleitner, Jakob H Macke, Sunil Puria (2024), From Simulations to Inference: Using Machine Learning to Tune Patient-Specific Finite-Element Models of the Middle Ear Towards Objective Diagnosis, *bioRxiv* [39]

Yves Bernaerts, **Michael Deistler**, Pedro J Gonçalves, Jonas Beck, Marcel Stimberg, Federico Scala, Andreas S Tolias, Jakob H Macke, Dmitry Kobak, Philipp Berens (2023), Combined statistical-mechanistic modeling links ion channel genes to physiology of cortical neuron types, *bioRxiv* [40]

Sebastian Bischoff, Alana Darcher, **Michael Deistler**,... (2024), A Practical Guide to Statistical Distances for Evaluating Generative Models in Science, *TMLR* [41]

Manuel Gloeckler, **Michael Deistler**, Christian Weilbach, Frank Wood, Jakob H Macke (2024), All-in-one simulation-based inference, *ICML (oral)* [42]

Jonas Beck, Nathanael Bosch, **Michael Deistler**, Kyra L Kadhim, Jakob H Macke, Philipp Hennig, Philipp Berens (2024), Diffusion Tempering Improves Parameter Estimation with Probabilistic Integrators for Ordinary Differential Equations, *ICML* [43]

Mila Gorecki, Jakob H. Macke, **Michael Deistler** (2024), Amortized Bayesian Decision Making for simulation-based models, *TMLR* [44]

Manuel Gloeckler, **Michael Deistler**, Jakob H Macke (2023), Adversarial robustness of amortized Bayesian inference, *ICML* [45]

Jonas Beck, **Michael Deistler**, Yves Bernaerts, Jakob H. Macke, Philipp Berens (2022), Efficient identification of informative features in simulation-based inference, *NeurIPS* [46]

Manuel Gloeckler, **Michael Deistler**, Jakob H. Macke (2022), Variational methods for simulation-based inference, *ICLR (spotlight)* [47]

Maximilian Dax, Stephen R. Green, Jonathan Gair, **Michael Deistler**, Bernhard Schölkopf, Jakob H. Macke (2022), Group-equivariant neural posterior estimation, *ICLR* [48]

Pedro J. Gonçalves†, Jan-Matthis Lueckmann†, **Michael Deistler**†, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F. Podlaski, Tim P. Vogels, David S. Greenberg, Jakob H. Macke (2020), Training deep neural density estimators to identify mechanistic models of neural dynamics, *Elife* [20]

† indicates shared first authorship.
‡ indicates shared last authorship.

2

BACKGROUND: FROM SIMULATION TO INFERENCE

> *Parameter inflation, we contend, is the Achilles' heel of compartmental [biophysical neuron] modeling.*
>
> — Almog & Korngreen [10]

In this chapter, we will provide background for the methods developed in this thesis. First, we will outline how biophysical models of neural dynamics are defined and how the differential equations are integrated. Second, we will describe the core machinery for fitting biophysical models to data: Statistical inference. Third, we will outline relevant background on simulation-based Bayesian inference methods with generative models. Finally, we will provide background on automatic differentiation and how it enables backpropagation of error for arbitrary software.

2.1 BIOPHYSICAL MODELS OF NEURAL DYNAMICS

Biophysical simulations model cellular and synaptic processes underlying observed measurements. In this section, we will describe the equations governing biophysical single-cell models and we will outline numerical routines that can solve the ordinary differential equations governing these systems.

2.1.1 *The Hodgkin–Huxley model*

Following Hodgkin & Huxley [1], biophysical neuron models are based on an electrical equivalent circuit of a neuron in which the membrane is modeled as a capacitor. Morphological detail is incorporated via the cable equation [2]. This leads to the following set of differential equations governing the evolution of voltage V at location x:

$$C_x \frac{\mathrm{d}V_x}{\mathrm{d}t} = i_x^{\text{mem}} + g_x^{\text{axial}} \cdot \frac{\mathrm{d}^2 V_x}{\mathrm{d}x^2}, \tag{2.1}$$

where C_x is the capacitance, i_x^{mem} is the current flowing through the membrane, and g_x^{axial} is the axial conductance at location x.

The membrane current i_x^{mem} is the sum of several membrane conductances. Each membrane conductance is (typically) modeled by a resistor and a battery. Therefore:

$$i_x^{\text{mem}} = \sum_{\text{channels}} g_x^{\text{channel}} (E_x^{\text{channel}} - V_x), \tag{2.2}$$

where g_x^{channel} is the channel conductance and E_x^{channel} is the reversal potential of the respective ion at location x. Many ion channels are active, meaning that their

conductance g_x^{channel} has a state that depends on voltage. This dependence can be expressed through gating variables, each of which is governed by a differential equation itself. For example, a typical model for potassium ion channels is

$$g_x^{\text{K}} = \overline{g}_x^{\text{K}} n_x^4, \tag{2.3}$$

$$\frac{\mathrm{d}n_x}{\mathrm{d}t} = \alpha(V_x)(1 - n_x) + \beta(V_x) n_x, \tag{2.4}$$

where we introduced the maximal conductance $\overline{g}_x^{\text{K}}$ and the gating variable $n_x \in [0, 1]$. The functions $\alpha(V_x)$ and $\beta(V_x)$ define opening and closing rates.

2.1.2 *Numerical solvers for Hodgkin–Huxley-type models*

Having described the differential equations that govern the membrane voltage, we now turn to how one can numerically integrate this equation. To do so, we will first spatially discretize (or "compartmentalize") the equations and then describe implicit Euler solvers that can numerically solve these differential equations.

2.1.2.1 *Spatial discretization*

One can spatially discretize Eq. 2.1 into

$$f(V_n) = \frac{\mathrm{d}V_n}{\mathrm{d}t} = \frac{1}{C_n} \cdot \left(i_n^{\text{mem}} + g_{n-1,n}^{\text{axial}}(V_{n-1} - V_n) + g_{n+1,n}^{\text{axial}}(V_{n+1} - V_n) \right), \tag{2.5}$$

where we have defined the vectorfield $f(V_n)$, and we have replaced the continuous location x with an integer n indicating the index of the compartment [2]. The equation above defines the vectorfield for an uninterrupted cable, branchpoints have to be dealt with separately [3].

2.1.2.2 *Implicit Euler solvers*

In principle, Eq. 2.5 can be solved with any differential equation solver. However, forward methods often fail to produce accurate simulations or can be unstable [11]. In particular short compartments or very strongly coupled compartments can lead to failure of forward methods (Fig. 2.1, red). Therefore, the voltage equations of biophysical neuron models (but not necessarily those of the gates) are typically solved with implicit Euler methods (or variants thereof), which are numerically more stable (Fig. 2.1, black) [11].

Implicit Euler methods discretize the differential equation in time as

$$V_n^\tau = V_n^{\tau+1} - \Delta\tau \cdot f(V_n^{\tau+1}), \tag{2.6}$$

where the superscript τ is a discrete time point and $\Delta\tau$ is the time step. Performing such an update requires to solve Eq. 2.6 for $V_n^{\tau+1}$. Generic routines to solve this equation would require an iterative procedure (e.g., Newton-Raphson). However,

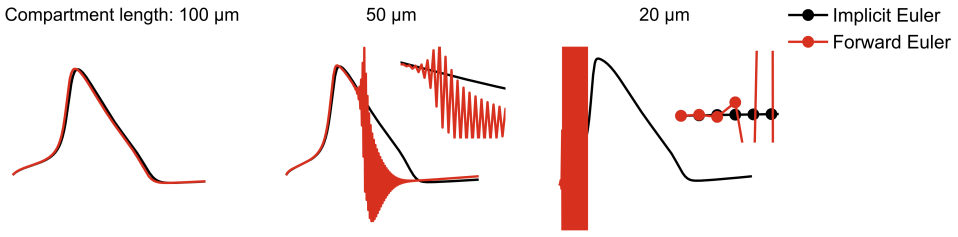


FIGURE 2.1: **Numerical stability of biophysical multicompartment models.** Simulations of a single branch consisting of four compartments of equal length. For sufficiently long compartments (left, 100 µm), forward Euler (red) is stable and produces a similar solution as Implicit Euler (black). For smaller compartments (middle, 50 µm) forward Euler becomes unstable. For even smaller compartments, forward Euler explodes after just a few timesteps (right, 20 µm). Implicit Euler is stable for all shown compartment sizes. Insets magnify the regions in which forward Euler becomes unstable.

for biophysical neuron models, the vectorfield f is linear *given the states of the gating variables*. Because of this, solving Eq. 2.6 turns into solving a linear system of equations. For unbranched cables, this linear system of equations is tridiagonal [49], whereas for branched cables, the system is quasi-tridiagonal. Such linear systems can be solved in $\mathcal{O}(N)$, where N is the number of compartments [11].

These equations and solvers are implemented in several toolboxes for biophysical simulation, most notably the NEURON toolbox [50].

2.2 INFORMATION FROM MEASUREMENTS: STATISTICAL INFERENCE

Having described how to set up biophysical neuron models, we now turn to *inferring* their free parameters from data. For biophysical neuron models, it is common that the maximal conductances $\overline{g}_x^{\text{channel}}$ cannot be measured and must be inferred. In some cases, opening and closing rates, the membrane capacitance C_x, or reversal potentials E_x^{channel} are not based on exact measurements and must also be inferred. To reduce the number of free parameters, it is sometimes assumed that these quantities have the same value in large parts of the neuron (i.e., they do not depend on the location x), although this assumption might not necessarily be true [51].

To infer properties of scientific simulators such as those in biophysics, we will rely on two methods for statistical inference: Maximum-likelihood estimation and Bayesian inference. Before introducing these two methods, we will first introduce how biophysical models can be described as statistical models, such that they are amenable to statistical inference.

2.2.1 *Biophysical models as statistical models: Noise models*

Biophysical models, described as ordinary, partial, or stochastic differential equations can be framed as statistical models by considering the simulator to represent

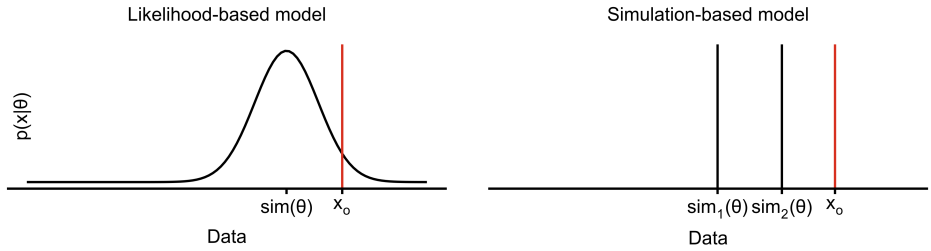


FIGURE 2.2: **Noise models and their effect on being able to evaluate the likelihood.** Left: Given any set of parameters, likelihood-based models can evaluate the likelihood of any data $p(\boldsymbol{x}|\boldsymbol{\theta})$ (including the observation $\boldsymbol{x}_o$). Right: Simulation-based models generate a sample from the likelihood $x \sim p(\boldsymbol{x}|\boldsymbol{\theta})$ (or multiple samples by simulating multiple times), but they do not necessarily allow likelihood evaluations.

the *likelihood* $p(\boldsymbol{x}|\boldsymbol{\theta})$, where we defined $\boldsymbol{\theta}$ as the free parameters and $\boldsymbol{x}$ as data (e.g., voltage response of the neuron). The likelihood describes how likely data $\boldsymbol{x}$ is given a particular parameter set $\boldsymbol{\theta}$. In addition to the differential equation governing the biophysical model, casting a biophysical model as a likelihood requires defining a *noise model*[1]. We will now turn to two illustrative and popular noise models for biophysical models.

The first noise model is an additive Gaussian observation noise model. Under the assumption that measurements are normally distributed samples around the true data and the simulated process itself (the mechanistic model) is not stochastic, the likelihood of any observation $\boldsymbol{x}_o$ is

$$p(\boldsymbol{x}_o|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}_o; \text{sim}(\boldsymbol{\theta}), \Sigma), \tag{2.7}$$

where we have introduced Σ as the covariance matrix describing the measurement process and $\text{sim}(\boldsymbol{\theta})$ as the simulation output given parameters $\boldsymbol{\theta}$. Under this noise model, the likelihood can be *evaluated* by running the simulator and evaluating $\boldsymbol{x}_o$ under a Gaussian distribution with the mean value being the simulation result and covariance Σ (Fig. 2.2, left). Being able to evaluate the likelihood enables the use of many popular parameter inference methods such as maximum-likelihood estimation, Markov-Chain Monte–Carlo (MCMC), or variational inference. The likelihood can also be *sampled* by running the simulator and then adding Gaussian noise with the covariance of the measurement process.

A second illustrative and popular noise model for biophysics is a current noise model [52]. To (phenomenologically) model inherent stochasticity of the cell response (e.g., stochasticity in channel opening and closing), this model adds noise to the input current of the neuron. This turns the partial differential equation governing the voltages in biophysical models into the *stochastic* partial differential equation

$$C_x \frac{\mathrm{d}V_x}{\mathrm{d}t} = i_x^{\text{mem}} + g_x^{\text{axial}} \cdot \frac{\mathrm{d}^2 V_x}{\mathrm{d}x^2} + \sigma \mathrm{d}B_t, \tag{2.8}$$

1 A deterministic (noise-free) simulation can be considered as a statistical model whose likelihood is a Dirac delta function at the simulation result.

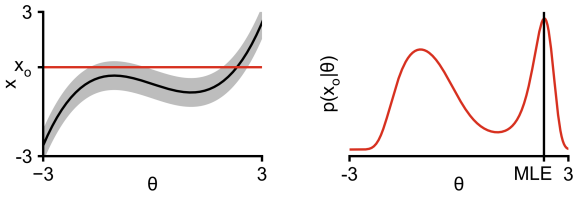


FIGURE 2.3: **Maximum likelihood estimation in a simulation with additive Gaussian noise.** Left: Deterministic simulation result given parameters (black) and standard deviation of measurement process (gray shaded). Right: The likelihood $p(\boldsymbol{x}_o|\boldsymbol{\theta})$ as a function of $\boldsymbol{\theta}$, evaluated as a horizontal cut at $\boldsymbol{x} = \boldsymbol{x}_o$ through the plot shown on the left. Maximum likelihood estimate (MLE) as black vertical line.

where B_t is a stochastic process describing the current noise. The likelihood of this model $p(\boldsymbol{x}_o|\boldsymbol{\theta})$ can, in principle, be evaluated by solving the Fokker-Planck-Equation, but even for moderately sized systems this is computationally infeasible. Sampling the likelihood, however, is typically possible by solving the stochastic (partial) differential equation with appropriate numerical routines[2] (Fig. 2.2, right). In order to generate multiple samples from the likelihood, the stochastic (partial) differential equation has to be solved multiple times, inducing additional computational cost.

How should one choose the noise model? Ideally, the noise model exactly matches the stochasticity believed to be at play in the studied system [52]. As this is often unknown, one can also fit parameters of the noise model to data [20]. Some noise models limit the range of methods that one can use to perform inference and, therefore, the choice of inference method can also influence the choice of noise model (at the cost of a potentially less faithful biophysical simulation).

The biophysical simulator, together with a noise model, defines the likelihood of simulation outputs given parameters $p(\boldsymbol{x}|\boldsymbol{\theta})$. Given this likelihood, we now turn to approaches for performing parameter inference given data. In the following two sections, we will review two popular inference approaches in biophysical models in neuroscience, as well as in many other disciplines in science and engineering: Maximum-likelihood estimation and Bayesian inference.

2.2.2 *Maximum-likelihood estimation*

Maximum-likelihood estimation returns those parameters which are most likely to have generated the dataset $\boldsymbol{x}_o$:

$$\boldsymbol{\theta}_{\mathrm{MLE}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{x}_o|\boldsymbol{\theta}). \tag{2.9}$$

To identify $\boldsymbol{\theta}_{\mathrm{MLE}}$, one has to search the parameter space. In models with few parameters (e.g., just a single one), this can be done by enumerating parameter values on a grid [54], running the simulation, and evaluating the likelihood given

2 Any numerical inaccuracies or uncertainties can lead to likelihood samples not being exact [43, 53].

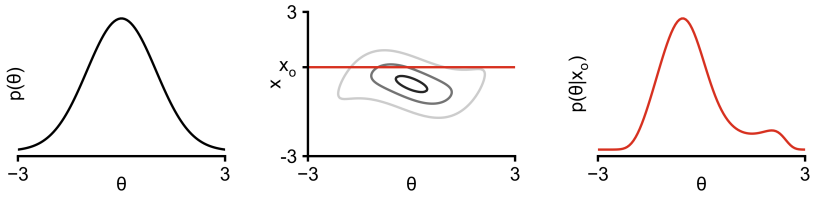


FIGURE 2.4: **Bayesian inference.** Left: Prior distribution $p(\boldsymbol{\theta})$ over parameters. Middle: Joint distribution $p(\boldsymbol{\theta}, \boldsymbol{x})$. Red line indicates cut through the joint distribution at observation $\boldsymbol{x}_o$. Right: Posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{x}_o)$, which can be obtained by evaluating the joint distribution along a cut at $\boldsymbol{x}_o$ and normalizing the resulting cut.

the observed data under the simulation result (Fig. 2.3). For models with many parameters, enumerating the possible values on a grid becomes infeasible (due to the curse of dimensionality) and, therefore, computing the MLE requires an efficient search method. If the simulation (and the noise model) are differentiable, gradient descent is a popular and powerful search method. A major limitation of maximum-likelihood estimation is that it returns only a single, "best", parameter set. This prevents studying the uncertainty of the parameter estimate, and it prohibits studying whether there are disparate parameter configurations that could match the data.

2.2.3 *Bayesian inference*

For many biophysical systems, many different parameter configurations can lead to similar neural activity [13]. This feature, ubiquitous in biology, is called *degeneracy* [55]. As we will show later, to study degeneracy, it is desirable that inference methods return the full space of parameters which match the data. One option to obtain this parameter space is to perform Bayesian inference.

Bayesian inference considers parameters and data as random variables. Inference is then performed by computing the conditional distribution of parameters given observed data

$$p(\boldsymbol{\theta}|\boldsymbol{x}_o) = \frac{p(\boldsymbol{x}_o|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{x}_o)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}. \tag{2.10}$$

The numerator of this fraction is the product of likelihood and prior, which can also be identified as the joint likelihood $p(\boldsymbol{\theta}, \boldsymbol{x}_o)$ evaluated at the observation. The denominator $p(\boldsymbol{x}_o)$, known as the evidence, describes how likely an observation $\boldsymbol{x}_o$ is on average across all parameter sets. Intuitively, the posterior distribution can be visualized as a cut through the joint density $p(\boldsymbol{\theta}, \boldsymbol{x})$ at $\boldsymbol{x} = \boldsymbol{x}_o$ and normalizing the resulting cut (by dividing by the evidence, Fig. 2.4).

Unlike maximum-likelihood estimation, Bayesian inference recovers the full space of parameters in the form of a posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{x}_o)$. In addition, the result of Bayesian inference depends on the prior distribution $p(\boldsymbol{\theta})$, which allows scientists to incorporate prior knowledge about parameters into the inference process.

A central challenge in Bayesian inference is that the denominator $p(\boldsymbol{x}_o)$ is intractable for simulators with many parameters because

$$p(\boldsymbol{x}_o) = \int p(\boldsymbol{x}_o|\boldsymbol{\theta})p(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}, \tag{2.11}$$

which, for simulators with M parameters, requires computing an M-dimensional integral. To overcome this, multiple methods have been developed, ranging from MCMC to variational inference [56, 57]. However, these methods require evaluations of the likelihood which, as we discussed above, are not necessarily available for all choices of noise models. In addition, even if the likelihood can be evaluated, these methods can be computationally expensive because they require repeated (often sequential) evaluations of the likelihood, each of which requires running a simulation. Simulation-based Bayesian inference methods have been developed specifically to overcome such limitations of traditional Bayesian inference methods.

2.3 SIMULATION-BASED BAYESIAN INFERENCE

Simulation-based Bayesian inference (often called simulation-based inference, or sometimes likelihood-free inference) performs Bayesian inference for simulators. If evaluating the likelihood is intractable, the joint distribution $p(\boldsymbol{\theta}, \boldsymbol{x}_o) = p(\boldsymbol{x}_o|\boldsymbol{\theta})p(\boldsymbol{\theta})$ cannot be evaluated, preventing the application of MCMC or variational inference. Instead, in order to compute (or approximate) the posterior distribution, one has to rely on samples from the likelihood for different parameter sets (Fig. 2.5). A set of methods to approximate the posterior in such scenarios is known as Approximate Bayesian Computation (ABC). ABC methods approximate the "cut" through the joint density by accepting samples from the joint distribution which are within an ϵ-region around the observation. For $\epsilon \to 0$, ABC converges to the true posterior, but will accept no (or very few) samples. As such, in practice, ABC typically requires $\epsilon > 0$, which induces an error in the posterior approximation.

Recently, several methods for *neural* simulation-based Bayesian inference methods have been developed. These methods train neural networks on simulated data and then evaluate the trained network on observed data to return the posterior distribution. Depending on the method, the neural network can approximate the posterior distribution (Neural Posterior Estimation, NPE [58]), the likelihood (Neural Likelihood Estimation [59]), the likelihood-to-evidence ratio (Neural Ratio Estimation, NRE [60, 61]), or arbitrary conditional distributions of the joint distribution [42]. NLE and NRE require an additional inference step to obtain the posterior (e.g., MCMC or variational inference [47]). As an illustrative example, we describe NPE below.

Given a prior $p(\boldsymbol{\theta})$ and a simulator that allows to draw samples from the likelihood $\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{\theta})$, NPE aims to identify the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{x}_o)$ given measured data $\boldsymbol{x}_o$. The core idea of NPE is that, if $\boldsymbol{\theta}, \boldsymbol{x} \sim p(\boldsymbol{\theta}, \boldsymbol{x})$, then a conditional density estimator $q_\phi(\boldsymbol{\theta}|\boldsymbol{x})$ trained on these data will, for every data $\boldsymbol{x}_o$, converge to the conditional distribution $p(\boldsymbol{\theta}|\boldsymbol{x}_o)$, which is the posterior [58]. For example,

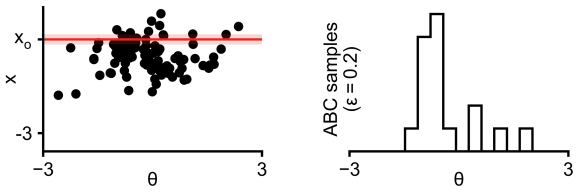


FIGURE 2.5: **Simulation-based Bayesian inference.** Left: Samples from the joint distribution, generated by sampling parameters from the prior and, for each of them, running the simulator (i.e., sampling the likelihood). Observation $\boldsymbol{x}_o$ in red. Shaded area is $\epsilon = 0.2$ around $\boldsymbol{x}_o$. Right: Parameter sets that are within ϵ-region around the observation. This method of generating *approximate* posterior samples is known as Approximate Bayesian Computation (ABC).

one can train a conditional normalizing flow $q_\phi(\boldsymbol{\theta}; \boldsymbol{x})$ by minimizing its negative log-likelihood

$$\mathcal{L} = -\mathbb{E}_{p(\boldsymbol{\theta},\boldsymbol{x})}[\log q_\phi(\boldsymbol{\theta}; \boldsymbol{x})] = -\iint p(\boldsymbol{\theta}, \boldsymbol{x}) \, \log q_\phi(\boldsymbol{\theta}; \boldsymbol{x}) \, \mathrm{d}\boldsymbol{\theta} \, \mathrm{d}\boldsymbol{x}. \tag{2.12}$$

Using $p(\boldsymbol{\theta}, \boldsymbol{x}) = p(\boldsymbol{\theta}|\boldsymbol{x})p(\boldsymbol{x})$ and rearranging terms leads to

$$\mathcal{L} = \int p(\boldsymbol{x}) \int p(\boldsymbol{\theta}|\boldsymbol{x}) \, \log q_\phi(\boldsymbol{\theta}; \boldsymbol{x}) \, \mathrm{d}\boldsymbol{\theta} \, \mathrm{d}\boldsymbol{x}, \tag{2.13}$$

in which the inner integral $\int p(\boldsymbol{\theta}|\boldsymbol{x}) \, \log q_\phi(\boldsymbol{\theta}; \boldsymbol{x}) \, \mathrm{d}\boldsymbol{\theta}$ is proportional to the Kullback-Leibler divergence $\mathrm{D}_{\mathrm{KL}}(p(\boldsymbol{\theta}|\boldsymbol{x})||q_\phi(\boldsymbol{\theta}; \boldsymbol{x}))$. Therefore, this loss function is minimized if and only if, for all $\boldsymbol{x} \in \mathrm{supp}(p(\boldsymbol{x}))$, $q_\phi(\boldsymbol{\theta}; \boldsymbol{x}) = p(\boldsymbol{\theta}|\boldsymbol{x})$.

In order to generate samples from the joint distribution $p(\boldsymbol{\theta}, \boldsymbol{x})$, NPE samples parameters from the prior distribution and, for each of them, runs the simulator (i.e., samples the likelihood). Importantly, if parameters are not drawn from the prior, then the implied joint distribution of samples $\boldsymbol{\theta}, \boldsymbol{x}$ changes and a conditional density estimator trained to estimate the conditional distribution of parameters $\boldsymbol{\theta}$ given $\boldsymbol{x}$ will not necessarily converge to the posterior distribution anymore[3] [58]. In particular, regions in parameter space that are oversampled (in comparison to the prior) in the generation of the training dataset will also be oversampled in the learned posterior distribution (Fig. 2.6). In many cases, however, it is desirable to sample parameters from a distribution that is not the prior. For example, if one aims to optimize posterior quality given a single observation (and with a limited simulation budget), then one would want to use active learning methods to select parameters that should be simulated. To allow for this, a series of algorithms have been developed which adapt the loss-function of NPE or perform post-hoc corrections to ensure that the neural network converges to the true posterior distribution even if parameters are not sampled from the prior distribution [58, 62, 63].

3 Notably, a conditional density estimator trained to estimate the likelihood of data $\boldsymbol{x}$ given $\boldsymbol{\theta}$ will converge to the true likelihood for any distribution of parameter samples. Because of this, Neural Likelihood

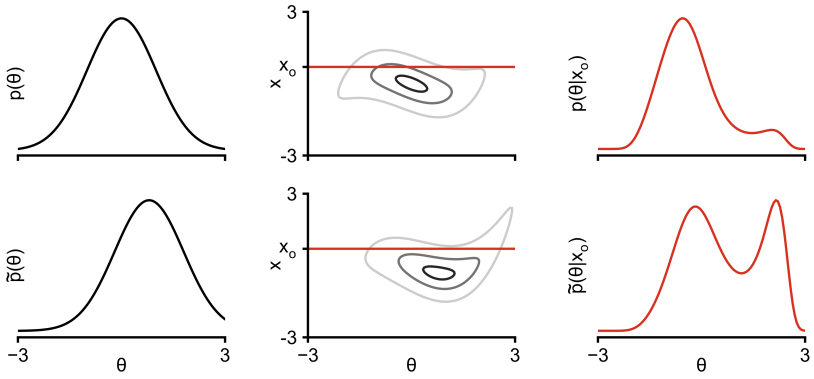


FIGURE 2.6: **Impact of parameters not following the prior distribution.** Top: Prior distribution (left) and the induced joint distribution (middle, contours). The conditional distribution of the joint is the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{x}_o)$ (right). Bottom: Sampling parameters from a proposal distribution $\tilde{p}(\theta)$ (left) that is not the prior leads to a different joint distribution (middle). This induces a different conditional distribution $\tilde{p}(\boldsymbol{\theta}|\boldsymbol{x}_o)$ (right).

Neural simulation-based Bayesian inference methods like NPE have several advantages over traditional Bayesian inference methods. First, they can run simulations fully in parallel and do not require sequential simulation steps. Second, unlike gradient-based Bayesian inference methods such as Hamiltonian Monte–Carlo or variational inference, they do not require gradients through the simulator. Third, like Approximate Bayesian Computation (ABC) methods, they only require samples from the likelihood (and no likelihood evaluations). Finally, the resulting methods are amortized: After an initial phase of simulation and training, the posterior distribution given any dataset can be obtained within (milli-)seconds with a single forward pass through the neural network.

2.4 AUTOMATIC DIFFERENTIATION & BACKPROPAGATION OF ERROR

Neural simulation-based Bayesian inference methods typically treat the simulation as a black-box, which makes the methods flexible and applicable to a wide range of simulators. This flexibility comes at a cost: By not including auxiliary information about the simulation such as gradients with respect to parameters, these methods can require many simulations and struggle with scaling to very high-dimensional parameter spaces [64].

Many simulators, especially in biophysical neuroscience modeling, are fully *differentiable* and permit evaluation of the gradient with respect to parameters. However, many computer programs such as biophysical simulations consist of a long sequence of computations, making it hard to evaluate the gradient analytically.

Estimation (NLE) and Neural likelihood-Ratio Estimation (NRE) do not require modifications of their loss function when active learning is used.

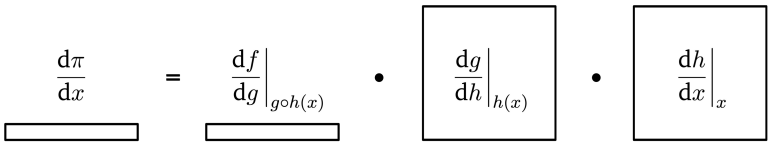


FIGURE 2.7: **Computational benefits of backpropagation of error.** Applying the chain rule to a sequence of operations requires multiplication of Jacobians. The squares (or rectangles) indicate the shapes of these Jacobians for a set of example operations. Multiplying Jacobians from right to left (forward-mode) requires a matrix-matrix multiplication, whereas multiplying Jacobians from left to right (reverse-mode, or backprop) requires only matrix-vector products.

Automatic differentiation overcomes this limitation by computing the gradient of any computer program. Consider a program π consisting of three sequential operations on the input x:

$$\pi(x) = f \circ g \circ h(x).$$

The gradient of π with respect to x can be evaluated via the chain rule:

$$\frac{\mathrm{d}\pi}{\mathrm{d}x} = \left.\frac{\mathrm{d}f}{\mathrm{d}g}\right|_{g\circ h(x)} \cdot \left.\frac{\mathrm{d}g}{\mathrm{d}h}\right|_{h(x)} \cdot \left.\frac{\mathrm{d}h}{\mathrm{d}x}\right|_{x}. \tag{2.14}$$

Differentiable programming languages track all operations occurring on x and implement the derivative of any such operation. This allows differentiable programming languages to use the chain rule to compute the gradient of any computer program.

Two methods are popular for evaluating the gradient via the equation defined by the chain rule in Eq. 2.14. The first of these methods multiplies the terms from right to left and is known as *forward-mode* automatic differentiation. For this method, the order of evaluating the gradient is the same as for evaluating the function $\pi(x)$ itself (i.e., first evaluating the gradient of h, then of g, then of f). The second method multiplies the terms from left to right and is known as *reverse-mode* automatic differentiation, or *backpropagation of error* (backprop) [65].

For some computer programs, backprop can be significantly more efficient than forward-mode automatic differentiation. To see this, consider the following example: $f : \mathbb{R}^N \to \mathbb{R}^1$, $g : \mathbb{R}^N \to \mathbb{R}^N$, $h : \mathbb{R}^N \to \mathbb{R}^N$, and $x \in \mathbb{R}^N$. In this case, the derivatives (Jacobians) in the chain rule of Eq. 2.14 are of shapes $\mathbb{R}^N$ for $\frac{\mathrm{d}f}{\mathrm{d}g}$, $\mathbb{R}^{N\times N}$ for $\frac{\mathrm{d}g}{\mathrm{d}h}$, and $\mathbb{R}^{N\times N}$ for $\frac{\mathrm{d}h}{\mathrm{d}x}$ (Fig. 2.7). When multiplying these terms from right to left, as would be done by forward-mode automatic differentiation, one has to compute a matrix-matrix product of two $N \times N$ matrices. When multiplying the terms from the left, however, all multiplications reduce to matrix-vector products. The latter can be significantly cheaper, especially for large matrices. This insight is particularly relevant for deep neural networks: These networks can have billions of parameters, but their output, the loss function, is typically a scalar value. Backprop ensures that the gradient of such programs is computationally cheap.

Across many fields such as particle physics or geophysics, differentiable simulators, in combination with backpropagation of error, have largely accelerated simulation-based parameter inference [27–31]. Unfortunately, prior to this thesis, no simulator existed for biophysics which was implemented in a differentiable programming language, preventing the use of gradient-based methods for simulation-based parameter inference.

3

SIMULATION-BASED BAYESIAN INFERENCE METHODS AND APPLICATIONS

Simulation-based Bayesian inference (SBI) is a powerful approach to perform inference in scientific simulators, in biophysics and beyond. Recent methods such as Neural Posterior Estimation (NPE) allow scientists to perform inference in black-box models with dozens of parameters and very high-dimensional simulation outputs. Crucially, by returning the full posterior distribution (as opposed to just a single "best" parameter set, as would be returned by maximum-likelihood estimation), SBI enables studies of parameter sensitivity or degeneracy.

Indeed, many systems in biology exhibit *degeneracy*: They can generate functionally similar behavior from disparate mechanisms. Degeneracy is believed to be crucial for natural selection, as it allows diverse solutions to exist, and it equips biological systems with robustness to perturbations of an underlying mechanism. Understanding which mechanisms make up for such robustness is a crucial challenge for biology. In a review from 2001, Edelman & Gally [55] wrote that

> further analysis [of degeneracy] will be particularly important in any attempt to deepen our understanding of biological complexity.

In cellular neuroscience, degeneracy has been observed in multiple experimental studies. For example, despite similar electrophysiological activity (Fig. 3.1, left), early measurements of ion channel densities of neurons in the crab *Cancer borealis* have yielded largely different values across animals [66] (Fig. 3.1, middle). SBI allows

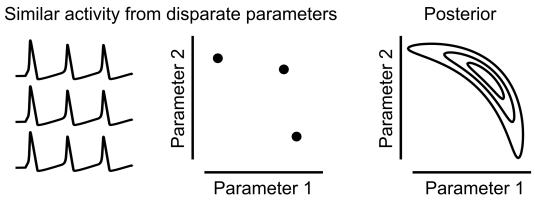


FIGURE 3.1: **Bayesian inference for studying degeneracy in neuroscience.** Different neural systems (e.g., different cells, or different animals) can show remarkably similar activity (left, three traces from three systems) although measurements of systems properties (i.e., parameters) reveal a large diversity (middle, one point for each system). We use simulation-based Bayesian inference to uncover the full space of data-compatible parameters in the form of the posterior distribution (right). All panels are illustrative and are not based on measurements.

scientists to study such degeneracy by uncovering the full space of data-compatible parameters in the form of the posterior distribution (Fig. 3.1, right).

Despite these benefits of SBI, at the time of me starting my PhD, it was challenging to use methods such as NPE in practice, for several reasons. First, active learning methods, necessary for simulation-efficient SBI, can be unstable when applied to real-world simulators. Second, many real-world measurements are not *exactly* compatible with any parameter set, making the simulator misspecified and leading to severe challenges for SBI [67]. Third, implementing SBI methods from scratch can be challenging and time-consuming for domain scientists, thus limiting the potential set of scientists who could benefit from these methods.

To overcome these limitations, we developed new methods for neural simulation-based Bayesian inference and used them to study degeneracy in neuroscience. Our advances improved the applicability, scalability, and robustness of SBI, and allowed us to perform inference on a series of models, ranging from a small circuit in the crab *Cancer borealis*, via a Hodgkin–Huxley (single compartment) model, to a biophysically detailed multi-compartment model of a layer 5 pyramidal neuron. Using the posterior distribution over parameters of the pyloric network in the crab *Cancer borealis*, we demonstrated that different parameter configurations which have virtually indistinguishable network activity might not necessarily be equally metabolically efficient.

In addition to these methods and applications of neural simulation-based Bayesian inference, over the past five years, we developed and maintained the `sbi` software toolbox. The `sbi` toolbox implements a range of popular algorithms for simulation-based Bayesian inference (covering NPE, NLE, and NRE methods) as well as tools for diagnosing the inference result and visualizing or analysing the posterior distribution. Over the past five years, the `sbi` toolbox has become a popular toolkit for scientists across many domains, and it has been used in fields ranging from particle physics [18], genetics [68], and economics [69] to astrophysics [23, 25].

3.1 ENERGY EFFICIENT NETWORK ACTIVITY FROM DISPARATE CIRCUIT PARAMETERS

Our paper "Energy efficient network activity from disparate circuit parameters" was published in the Proceedings of the National Academy of Sciences (PNAS) [32]. An earlier version of this work was made available as a preprint [70].

Disparate energy consumption despite similar network activity

The pyloric network in the crab *Cancer borealis* controls muscles which in turn control parts of its digestive tract [71]. The network activity of this circuit can be modeled with three model neurons connected by seven inhibitory synapses (Fig. 3.2a). In this model, each neuron contains a variety of ion channels, leading to a total of 31 parameters (eight parameters for each of the three neurons, seven synaptic parameters).

The seminal results by Prinz, Bucher & Marder [13] demonstrated that, in a model of the pyloric network in the crab *Cancer borealis*, different circuit configurations could lead to remarkably similar network activity (Fig. 3.2b). In Gonçalves *et al.* [20], we had reproduced this finding with Neural Posterior Estimation (NPE): The

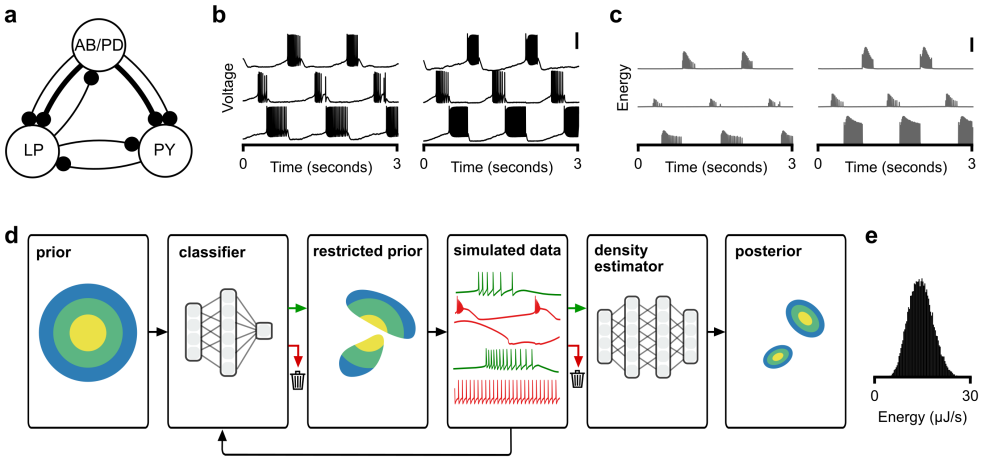


FIGURE 3.2: **Similar activity with different energy consumption.** (a) Computational model of the pyloric network consisting of three model neurons (AB/PD, LP, PY) and seven synapses. (b) Two model configurations with similar circuit activity (traces from top to bottom: AB/PD, LP, PY) despite different circuit parameters (parameter values not plotted). Scale bars indicate 50 mV. (c) Consumed energy at each time point. Scale bar indicates 100 μJ/s. (d) Inferring the posterior distribution by combining a rejection classifier and a deep neural density estimator. (e) Histogram over energy consumed by the circuit, divided by the duration of the simulation. Trace with lowest energy consumes nine times less energy than trace with highest energy. Figure adapted from Deistler, Macke & Gonçalves [32].

posterior marginals covered almost the entire prior space, thereby allowing for large deviations of individual parameters. Upon inspecting posterior predictive samples, we then made a surprising discovery: Although different parameters all had similar activity patterns, the energy consumption of these solutions could be largely different (Fig. 3.2c) [72].

Given the observation that two circuit configurations could have disparate energy consumption despite similar network activity, we quantified *how large* this disparity in energy consumption could be. To this end, we aimed to generate a database of parameter sets which have virtually identical network activity. Doing this is challenging, even with powerful inference methods such as NPE: First, NPE requires many simulations because a large fraction of prior samples does not generate sensible network activity. Second, even if one managed to identify the true posterior distribution, the resulting posterior predictive samples would not necessarily match the observation due to stochasticity in the simulator, which could bias an analysis of the range of energy consumptions. To overcome these limitations, we extended NPE in two ways: First, we trained an additional classifier to constrain the prior to regions that are more likely to generate good parameter sets (Fig. 3.2d). Second, we post-hoc filtered 1M posterior predictive samples to be within an ϵ-ball around the observation[1]. By combining these two advances, we generated a database of 35k parameter sets, all of which closely matched experimental measurements. We then computed the energy consumption induced by each of these parameter sets. Despite their similarity in network activity, the parameter sets could differ by an order of magnitude in their metabolic cost (Fig. 3.2e).

This observation raised several questions: How much degeneracy remains when low energy consumption is enforced? Which parameter values are required to achieve low energy consumption? How can low energy consumption be achieved on a circuit level? How does low energy consumption trade-off with other requirements such as robustness to changes in temperature?

Metabolic constraints on individual circuit parameter ranges

We studied how strongly parameter degeneracy would be constrained by enforcing low energy consumption. Given that the energy consumption between all data-matching parameter sets could vary by a factor of nine, one might speculate that enforcing low energy consumption largely reduces degeneracy. However, by a simple statistical argument, this is *not* expected: In a model with 31 parameters, reducing the volume of the parameter space by a factor of 50 (by enforcing energy consumption to be in the lowest 2 % quantile) is expected to reduce the degeneracy of every individual parameters by only 12 % ($1 - 0.02^{1/31}$). We tested this hypothesis in our model. Among our database of 35k data-matching parameters, we selected the ones in the 2 % quantile of lowest (or highest) energy consumption (Fig. 3.3a).

1 This additional step requires 1M additional simulations. In addition, the accepted samples are no longer samples from the exact posterior distribution (even if the NPE samples were exact posterior samples). To highlight this, we call these samples "data-matching samples", but avoid the term "posterior samples".

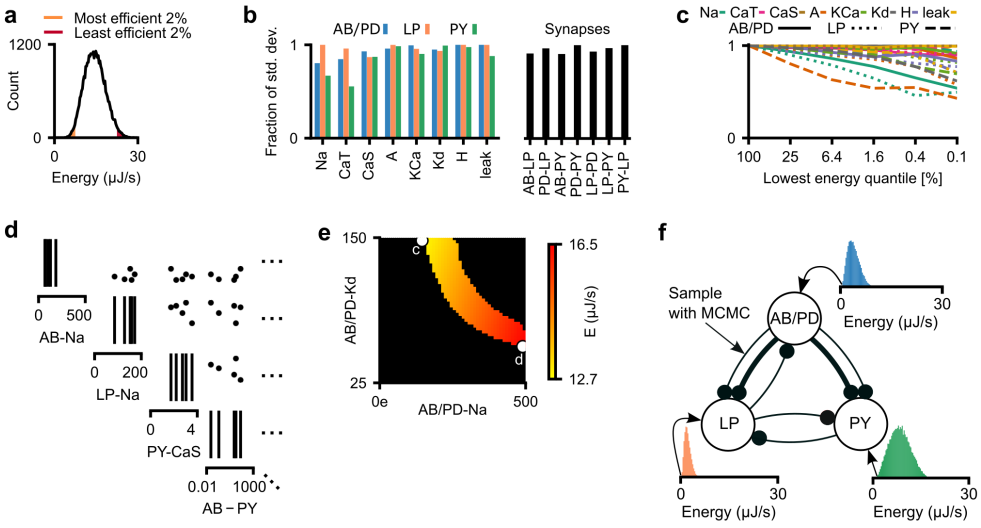


FIGURE 3.3: **Metabolic constraints on individual circuit parameters.** (a) Time-averaged energy consumption of models that match experimental data. Orange area is to the energy consumption in the lowest 2% quantile, red area in the top 98% quantile. (b) Standard deviation of parameters for models with energy consumption in the lowest 2% quantile. Standard deviation is normalized to the standard deviation of the parameters across all 35,939 models in our database. (c) Same as panel (b), but for a range of quantiles. Solid: AB/PD, dotted: LP, dashed: PY. (d) Subset of the parameter values of the five most efficient circuit configurations in our database. (e) Time-averaged energy consumption (as predicted by linear regression) of several models that differ only in their maximal conductances of Na and Kd in the AB/PD neuron. (f) We select the five most efficient parameter sets for each neuron separately, and search with MCMC for synaptic conductances such that the target circuit activity is achieved. Figure adapted from Deistler, Macke & Gonçalves [32].

Despite this additional constraint, many parameters were degenerate and some parameters could still vary by an order of magnitude. We quantified this degeneracy by comparing the fraction of the posterior (marginal) standard deviation of all parameters before and after constraining the configurations to be in the 2 % lowest energy quantile (Fig. 3.3b). Sodium and transient and slow calcium maximal conductances were most strongly constrained by enforcing low energy consumption, but significant degeneracy remained even for those conductances. We then constrained the metabolic cost further but found that this impacted the remaining degeneracy only weakly (Fig. 3.3c). Indeed, even the five most efficient circuit configurations in our database of 35k parameter sets still had largely disparate parameter values (Fig. 3.3d).

Following this analysis, we studied minimal perturbations which could maximally reduce energy consumption. By modifying just two parameters in a coordinated fashion, energy consumption could be largely reduced (for some pairs of parameters by up to 60 %, Fig. 3.3e). Finally, we studied whether neurons in the circuit could

be tuned individually for low energy consumption. We used the most efficient single neuron configurations in our database of 35k neurons and sampled synaptic conductances with Markov-chain Monte–Carlo (MCMC), such that the resulting circuit would match experimental activity (notably, thanks to the NPE posterior being able to evaluate the posterior density, this did not require any additional simulations, Fig. 3.3f). The resulting network configurations closely matched the recorded network activity, but their energy consumption was more efficient than that of any other configuration in our database of 35k neurons. This indicates that neurons could tune their parameters independently to achieve minimal energy consumption (only requiring to adjust synaptic conductances).

Robustness to temperature does not require an increased metabolic cost

In addition to functional circuit activity, the crab *Cancer borealis* also requires its nervous system to be robust to fluctuations in temperature to survive [74–77]. The water temperature in which crabs live can vary largely—which in turn changes the biochemical properties of its nervous system. We wondered whether we could find circuit configurations which match experimentally measured network activity, are robust to changes in temperature, and are metabolically cheap.

To study this, we further constrained the posterior distribution of NPE by additional recordings of the network activity at a higher temperature (Fig. 3.4a,b) [73]. The resulting parameter sets matched experimentally observed network activity at 11 °C and at 27 °C (Fig. 3.4c). We then studied the energy consumption of the resulting parameter sets. We found that energy consumption at 27 °C was correlated to the energy consumption at 11 °C, but did not yet determine the energy consumption uniquely, suggesting that different configurations could be more or less desirable at different temperatures (Fig. 3.4d). However, even when constraining circuit configurations to match data at 11 °C and at 27 °C, and to have energy consumption in the lowest 2 % quantile at both temperatures, the circuit parameters exhibited significant parameter degeneracy (Fig. 3.4e). Lastly, we studied a potential trade-off between temperature robustness and energy consumption. We computed the energy consumption of the posterior given recordings at 11 °C and 27 °C and compared it to the energy consumption of the posterior given recordings at 11 °C. Remarkably, the distributions were similar, or (for some experimental recordings) even showed a shift towards more energy-efficient parameters when temperature robustness was required (Fig. 3.4f). Overall, these results indicate that temperature robustness does not preclude energy efficiency.

Discussion

We extended and used NPE to identify data-matching parameters of a simulator of the pyloric network in the crab *Cancer borealis*. By studying the posterior distribution, we demonstrated that disparate parameter configurations could be metabolically

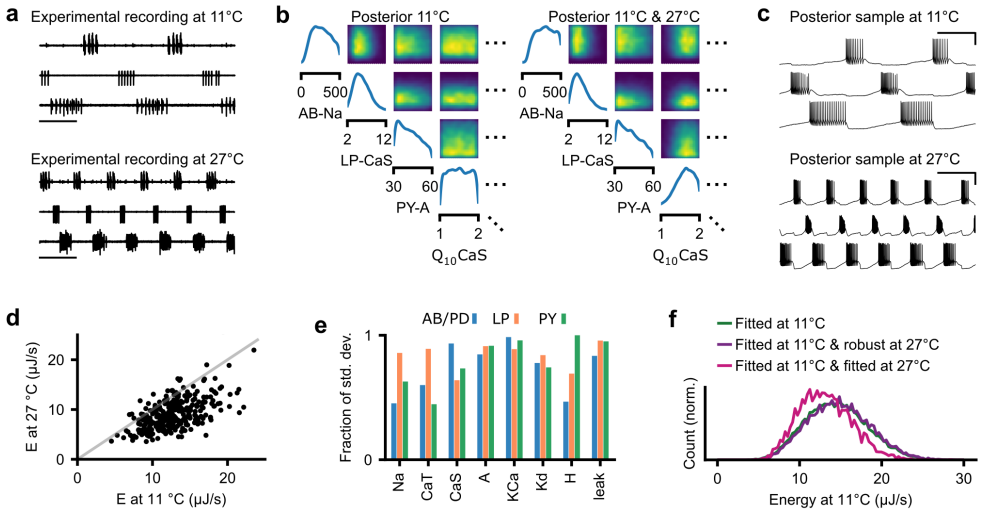


FIGURE 3.4: **Temperature robustness does not preclude energy efficiency.** (a) Top: Experimental data at 11 °C. Bottom: Experimental data at 27 °C [73]. (b) Left: Posterior distribution given experimental data at 11 °C. Right: Posterior given experimental data at 11 °C and 27 °C. (c) Simulations for a parameter set drawn from the posterior distribution matching experimental data at 11 °C and 27 °C. Simulations at 11 °C (top) and 27 °C (bottom). (d) Energy consumption at 11 °C versus 27 °C (time averaged) for 967 circuits sampled from the posterior (in (b) right). In grey, the identity line. (e) Standard deviation of parameters for models that match experimental data at 11 °C and 27 °C and that have energy consumption in the lowest 2% quantile at 11 °C and 27 °C. Standard deviation is normalized to the standard deviation of the parameters across all 35,939 models in our database. (f) Green: Distribution of the energy consumption of circuits matching experimental data at 11 °C. Purple: Distribution of the energy consumption of circuits that match data at 11 °C and are robust at 27 °C. Pink: Distribution of the energy consumption of circuits that match experimental data at 11 °C and 27 °C. Figure adapted from Deistler, Macke & Gonçalves [32].

cheap and robust to fluctuations in temperature. This allows disparate solutions to exist in nature, which has been argued to be crucial for natural selection and robustness [55].

In hindsight, this study—following up on our analyses in Gonçalves *et al.* [20]—formed one of the first demonstrations that Neural Posterior Estimation (NPE) could be used to gain scientific insight into neuroscience models. All analyses above were enabled by the property of NPE (and Bayesian inference in general) to return the full space of parameters that are compatible with data, thereby allowing us to study the full range of energy consumptions of data-compatible models. This property of NPE distinguishes it from other parameter fitting methods such as maximum-likelihood estimation or genetic algorithms: These methods typically return just a single—or few—data-matching parameter sets. Since the initial publication of this manuscript, several neuroscience studies have used NPE to infer the parameter posterior, which

has enabled neuroscientists to gain new insights into plasticity rules [78, 79], genetics [40], animal vision [80], neural connectivity patterns [81], or brain waves [38, 82].

3.2 TRUNCATED PROPOSALS FOR SCALABLE AND HASSLE-FREE SIMULATION-BASED BAYESIAN INFERENCE

Our paper "Truncated proposals for scalable and hassle-free simulation-based inference" was presented at the 36th Conference on Neural Information Processing Systems (NeurIPS) and was published in the conference proceedings [33]. An earlier version of this work was made available as a preprint [83].

Real-world applicability issues of previous active learning methods

In order to infer the posterior distribution, Neural Posterior Estimation (NPE) draws parameters from the prior. This has the advantage that the resulting posterior distribution is amortized (i.e., can be evaluated for any observation $\boldsymbol{x}_o$), but it comes at the cost of requiring a large number of simulations [84]. For example, in Deistler, Macke & Gonçalves [32], we had applied NPE to a simulator of the pyloric network in the crab *Cancer borealis*. Successfully inferring the posterior distribution came at a large computational cost: 9 million simulations of 3 seconds each, corresponding to 7,500 CPU hours (which is about ten CPU months). To remedy this cost, we had also evaluated *sequential* NPE (SNPE). These methods employ active learning to reduce the number of required simulations. However, by adaptively sampling parameters, SNPE methods have to modify the loss function or perform post-hoc correction steps. These modifications led all existing methods for SNPE to fail: SNPE-A [58] led to non-positive definite covariance matrices, SNPE-B had poor performance (likely due to high variance of the loss function) [62], and SNPE-C placed a large fraction of mass outside of the prior bounds [63]. To overcome these limitations, we developed a new algorithm for SNPE, which we called Truncated Sequential Neural Posterior Estimation (TSNPE). Using TSNPE, we were able to successfully infer the posterior of the pyloric network with only 390k simulations, more then an order of magnitude less then previous NPE methods. In addition, we used TSNPE to perform inference in a biophysically detailed model of a layer 5 pyramidal cell, which had been inaccessible to previous methods.

TSNPE enables active learning and maximum likelihood training

TSNPE rests on a simple observation: The posterior will be the same for all prior distributions which are proportional to each other *at least on the support of the posterior*. This implies that, if the support of the posterior distribution is smaller than the support of the prior, one will obtain the same posterior for a prior which places zero mass outside of the posterior support.

Inspired by previous results by Blum & François [85], we aimed to use this observation for improving active learning for NPE. Assume, for now, that we know the support of the posterior. Then, instead of drawing parameters from the prior (as is done by NPE), we could train NPE on parameters drawn from a truncated prior

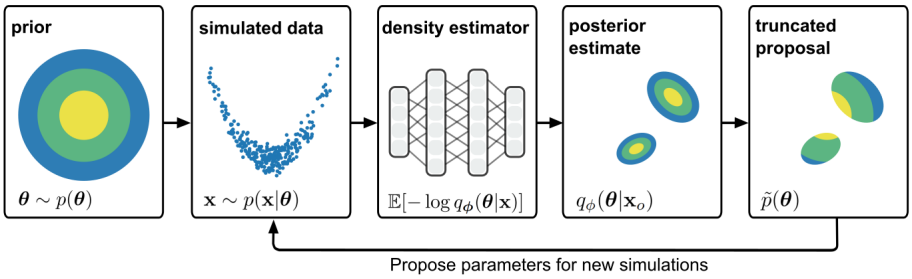


FIGURE 3.5: **Truncated Sequential Neural Posterior Estimation (TSNPE).** The method starts by sampling from the prior, running the simulator, and training a neural density estimator with maximum-likelihood to approximate the posterior. In subsequent rounds, parameters are sampled from the prior, but rejected if they lie outside of the support of the approximate posterior. With these proposals, the neural density estimator can be trained with maximum-likelihood in all rounds. Figure copied from Deistler, Goncalves & Macke [33].

[86], which is proportional to the prior on the support of the posterior and zero everywhere else. Training the NPE network on these simulations with maximum likelihood would converge to the same posterior distribution as training the neural density estimator on simulations generated from prior samples[2] (Fig. 3.5). However, because parameters are drawn from a truncated prior, the simulation results will typically be much closer to the observation $\boldsymbol{x}_o$, which we expected would improve simulation efficiency.

As such, we are left with one central challenge: To estimate the support of the posterior distribution. We achieve this by first using NPE to provide an approximate posterior distribution, and then use its 1-ϵ highest-probability region (HPR_ϵ, we used values for ϵ around 10^{-4}) as proxy for the posterior support. Since NPE is trained with a maximum-likelihood loss, its posterior estimate is expected to be mode-covering, and thus we expected to obtain an HPR_ϵ that is at least as broad as the HPR_ϵ of the true posterior. We note that, while using $\epsilon > 0$ systematically neglects a part of the posterior support, we will show empirically that this error is by far outweighed by other factors (e.g., suboptimal convergence of the neural network).

To summarize, TSNPE proceeds as follows: We first sample from the prior, run simulations, train the NPE network, and infer the posterior given an observation $\boldsymbol{x}_o$. We then use its HPR_ϵ as approximate posterior support. We draw samples from the prior, reject those which were outside of the approximate posterior support, run simulations, and retrain the NPE network. We perform this procedure over multiple rounds (Fig. 3.5).

2 Note that this only holds for the observation within whose posterior support the parameters are drawn.

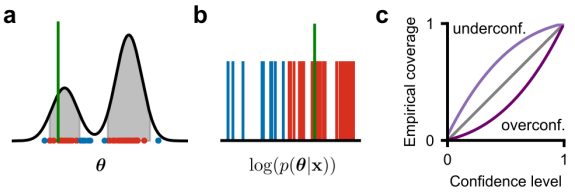


FIGURE 3.6: **Diagnostic tool.** (a) Parameter θ^* (green) lies within the 1-α confidence region (gray) of the estimated posterior. (b) $\log(p(\theta^*|x))$ is above the 1-α quantile of posterior samples. (c) 1-α versus empirical coverage, averaged over θ^*. Figure copied from Deistler, Goncalves & Macke [33].

A scalable diagnostic tool that evaluates expected coverage

The success of TSNPE rests on the assumption that the approximate posterior will cover the support (or at least the HPR_ϵ) of the true posterior. To diagnose potential issues in this, we suggested a sample-based method to estimate the expected coverage of TSNPE. This diagnostic tool uses the ability of NPE methods to sample *and* evaluate the approximate posterior. We showed that these two properties can be used to efficiently estimate expected coverage of the approximate posterior (Fig. 3.6), which can be used to diagnose whether the approximate posterior is underconfident or overconfident [86–91]. In the latter case, TSNPE might reject relevant parameter regions and requires measures to improve coverage, such as using neural network ensembles for training [92].

While we developed this diagnostic method for TSNPE, it is applicable to any method which returns an approximate posterior estimate whose log-probability can be evaluated up to a normalization constant. Because of this, the tool is also applicable to NPE, NLE, and NRE.

TSNPE enables efficient and hassle-free inference on real-world tasks

We evaluated TSNPE on a series of benchmark tasks [84]. We were particularly interested in two things: First, whether TSNPE would match the simulation efficiency of other SNPE methods and second, whether, empirically, the approximate posterior HPR_ϵ would cover the true posterior support.

Across six benchmark simulators, TSNPE was approximately equally simulation efficient as SNPE (being more efficient on some tasks and less efficient on others, Fig. 3.7, left), and both algorithms were significantly more efficient than amortized NPE. For all tasks, the approximate posterior HPR_ϵ was narrower than the prior, sometimes by several orders of magnitudes (Fig. 3.7, middle). In addition, for most tasks, and in particular for larger simulation budgets, the approximate posterior HPR_ϵ included all ground truth posterior samples. In some cases, the TSNPE posterior did not cover a large fraction of ground truth posterior samples. In these

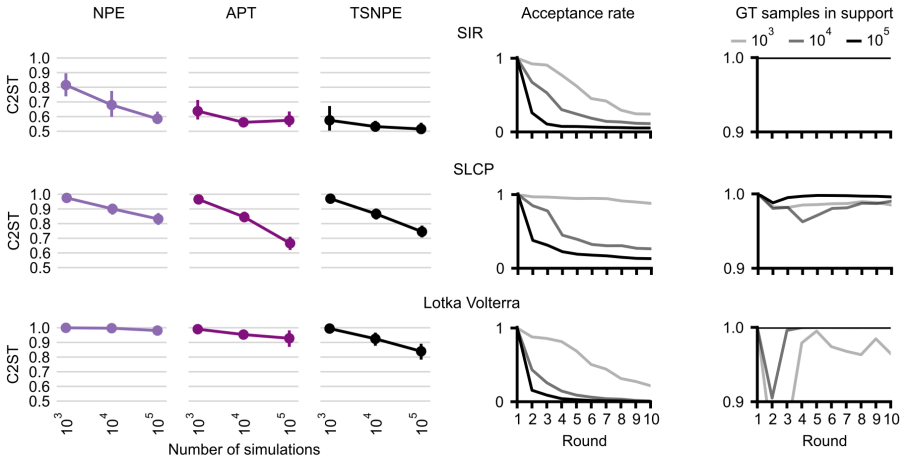


FIGURE 3.7: **Performance on benchmark tasks.** Left three columns: Classifier two-sample test accuracy (C2ST) of NPE (left), APT (middle), and TSNPE (right) for three simulation budgets. Forth column: Fraction of prior samples within the approximate-posterior HPR_ϵ in each round for each simulation budget. Fifth column: Fraction of true-posterior samples within the approximate-posterior HPR_ϵ. TSNPE with $\epsilon = 10^{-4}$ and rejection sampling from truncated proposal. Three additional benchmark tasks in Deistler, Goncalves & Macke [33]. Figure adapted from Deistler, Goncalves & Macke [33].

cases, however, all methods (NPE, SNPE, and TSNPE) had poor performance, suggesting that these tasks require a significantly larger simulation budget.

Having demonstrated that TSNPE matches the accuracy of state-of-the-art methods on benchmark tasks, we turned to the original motivation for developing TSNPE: To enable inference in real-world simulators with many parameters, on which current SNPE methods fail. To demonstrate that TSNPE enables posterior inference in these models, we applied it to two simulators: The pyloric network simulator from the crab *Cancer borealis* and a biophysically detailed multi-compartment model of a layer 5 pyramidal neuron.

For the pyloric network simulator, we applied TSNPE across 13 rounds. TSNPE successfully identified the posterior distribution given an experimental recording from this circuit (Fig. 3.8a) [73] with 390k simulations—40 times fewer than NPE (Fig. 3.8b) [20]. Previous SNPE methods failed on this task.

We then applied TSNPE to a simulator of the voltage response of a layer 5 pyramidal cell (Fig. 3.8c) [12, 93, 94]. We had attempted to infer the posterior for this simulator several times in the years before we developed TSNPE, but had never been successful: Amortized methods were not sufficiently simulation-efficient to converge in reasonable time and sequential methods failed due to the above-described issues. In contrast, TSNPE successfully inferred the posterior distribution within six rounds of simulations. The resulting posterior samples closely matched the observed voltage traces (Fig. 3.8b, additional samples in Appendix of Deistler, Goncalves & Macke [33]).

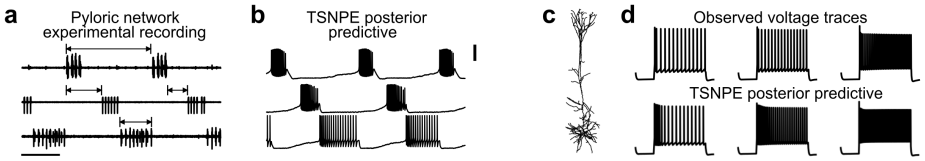


FIGURE 3.8: **Bayesian inference for two neuroscience simulators.** (a) Experimental recording from the pyloric network of the crab *Cancer borealis* [73]. Arrows indicate summary statistics. (b) TSNPE posterior predictive sample matches summary statistics of the experimental data. (c) Cell morphology of a layer 5 pyramidal cell. (d) Top: Synthetic observed voltage traces given three sets of stimuli. Bottom: Posterior predictive sample obtained with TSNPE. Figure modified from Deistler, Goncalves & Macke [33].

Discussion

The development of TSNPE was driven by unsatisfactory results of previous methods when employed on real-world tasks. On such tasks, these methods did not just perform poorly, but they failed catastrophically—forcing us to fall back on amortized neural posterior estimation without any active learning strategy, at the cost of requiring millions of simulations.

With TSNPE, we aimed to develop a method that is not necessarily more efficient than previous methods—indeed, our benchmark results showed that TSNPE does not outperform previous SNPE methods in terms of simulation efficiency. Instead, we aimed to build a method that is easy to run, reliable, and hassle-free. The ability of TSNPE to perform active learning while allowing to train with maximum likelihood provides these features, and our results on performing inference on two neuroscience examples demonstrate that TSNPE performs well on real-world tasks.

Since its publication in November 2022, TSNPE has been adopted to perform inference in several domains, ranging from astrophysics [95–97] and computational biology [19] to simulators of developmental EEG maturation [98]. In addition, the presented diagnostic tool has been used to diagnose potential mis-calibration in models from cosmology [99] and neuromorphic computing [100].

3.3 GENERALIZED BAYESIAN INFERENCE FOR SCIENTIFIC SIMULATORS VIA AMORTIZED COST ESTIMATION

Our paper "Generalized Bayesian Inference for Scientific Simulators via Amortized Cost Estimation" was presented at the 37th Conference on Neural Information Processing Systems (NeurIPS) and was published in the conference proceedings [101]. An earlier version of this work was made available as a preprint [102].

The limited flexibility of "standard" Bayesian inference

In Deistler, Macke & Gonçalves [32] we demonstrated how we used NPE to generate a database of parameter sets which match experimental recordings. In order to not bias any analyses of metabolic cost, we performed a post-hoc filtering step in which we rejected parameter sets whose simulation result was outside of an ϵ-ball around the observation. Because of this post-hoc filtering step, however, the database of parameter configurations no longer contained samples from the true posterior distribution. This made us wonder: Is it always desirable to obtain the Bayesian posterior distribution? Or do different studies, for example of degeneracy, require different targets for inference?

The starting point of our paper on "Generalized Bayesian inference for Scientific Simulators via Amortized Cost Estimation" was that we were not satisfied with the flexibility of Bayesian inference for studying degeneracy in neuroscience. Following our insights on the limitations of Bayesian inference for studying energy consumption, we quickly identified other cases in which exact Bayesian inference is not desirable. For example, if the simulator is *misspecified* (i.e., for no set of parameters can the simulator match the observation exactly) [103], then the evidence $p(\boldsymbol{x}_o)$ is zero, preventing the application of Bayesian inference. In these scenarios, it could still be desirable to have an algorithm which returns parameter sets whose predictive samples are close to the observation. Another example is the property of the posterior distribution to exclude parameters which can be close to the data, but never match it *exactly*. For an analysis of degeneracy, one might well be interested in including these parameter sets.

Generalized Bayesian inference [104] refers to a set of methods which improve the flexibility and applicability of Bayesian inference. Generalized Bayesian inference aims to identify the *generalized* posterior distribution

$$p(\boldsymbol{\theta}|\boldsymbol{x}_o) \propto \exp(\beta \cdot \ell(\boldsymbol{\theta}; \boldsymbol{x}_o)) p(\boldsymbol{\theta}), \tag{3.1}$$

with cost function $\ell(\boldsymbol{\theta}; \boldsymbol{x}_o)$ and hyperparameter β. If the cost function $\ell(\boldsymbol{\theta}; \boldsymbol{x}_o)$ is chosen to be the log-likelihood function and $\beta = 1$, then generalized Bayesian inference falls back on standard Bayesian inference. For other cost functions, however, it can flexibly incorporate any desiderata about the inference result into the inference process.

Performing generalized Bayesian inference for simulation-based models is challenging. To see this, consider a cost function of the form

$$\ell(\boldsymbol{\theta}; \boldsymbol{x}_o) \equiv \mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta})}[d(\boldsymbol{x}, \boldsymbol{x}_o)] = \int_{\boldsymbol{x}} d(\boldsymbol{x}, \boldsymbol{x}_o) p(\boldsymbol{x}|\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{x}. \tag{3.2}$$

Estimating the cost requires to run multiple forward simulations to approximate the expectation with a Monte–Carlo average. This is computationally expensive for expensive simulators, but it can also be inaccurate if too few Monte–Carlo simulations are used. To overcome these limitations, we developed a new method based on neural networks [34].

Amortized cost estimation for generalized Bayesian inference

To use neural networks for generalized Bayesian inference, we restricted ourselves to cost functions of the form:

$$\ell(\boldsymbol{\theta}; \boldsymbol{x}_o) \equiv \mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta})}[d(\boldsymbol{x}, \boldsymbol{x}_o)] = \int_{\boldsymbol{x}} d(\boldsymbol{x}, \boldsymbol{x}_o) p(\boldsymbol{x}|\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{x}. \tag{3.3}$$

Many popular cost functions can be written in this way: Mean squared error [104], mean absolute error, or maximum mean discrepancy (MMD) [105]. Our core insight is that we can learn to regress onto such cost functions by leveraging the property of neural network regression (trained with mean squared error) to converge onto the conditional expectation. After training, the neural network can be used to predict the cost of any given parameter set $\boldsymbol{\theta}$, and can thus be plugged into (doubly-intractable) MCMC schemes to generate samples from the generalized posterior.

Our algorithm proceeds as follows: We generate a simulated database of parameters $\boldsymbol{\theta}$ and simulation results $\boldsymbol{x}$, compute the distance from simulation results to the observation $d(\boldsymbol{x}, \boldsymbol{x}_o)$, and then train a neural network to predict this distance given the simulated parameters. In Gao, Deistler & Macke [34], we proved that a mean-squared error loss function (for an infinitely large dataset) is minimized if and only if the neural network predicts the true cost $\ell(\boldsymbol{\theta}; \boldsymbol{x}_o)$ for all parameters $\boldsymbol{\theta}$. On top of this algorithm, we also developed an *amortized* version which additionally conditions the neural network on the observation $\boldsymbol{x}_o$. We termed this method "Amortized Cost Estimation" (ACE).

Hodgkin–Huxley inference from Allen Cell Types Database recordings

We first applied ACE to a set of benchmark tasks. Across a broad range of cost functions, simulators, and observations, ACE could recover data-matching parameter sets. We then evaluated ACE on misspecified observations and found that it identified parameter samples that were significantly closer to the observation than state-of-the-art simulation-based Bayesian inference methods.

Next, we aimed to apply ACE to infer parameters of a Hodgkin–Huxley model given intracellular recordings from the Allen Cell Types Database (Fig. 3.9a) [106–

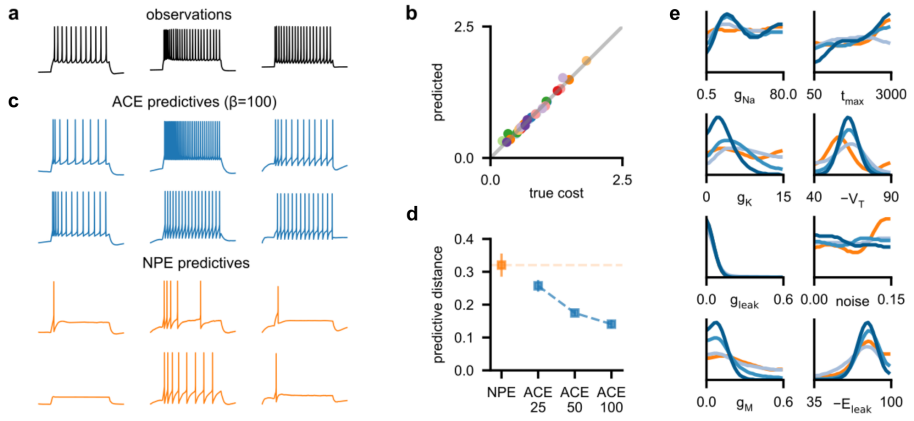


FIGURE 3.9: **Application of ACE to Allen data.** (a) Three observations from the Allen Cell Types Database. (b) True cost (evaluated as Monte-Carlo average over 10 simulations) per $\boldsymbol{\theta}$ vs ACE-predicted cost. Colors are different observations. (c) Marginals of posterior distributions for NPE (orange) and ACE (shades of blue. Light blue: $\beta = 25$, medium blue: $\beta = 50$, dark blue: $\beta = 100$). (d) Top: Two GBI predictive samples for each observation. Bottom: Two NPE predictive samples. Additional samples in Gao, Deistler & Macke [101]. (e) Average predictive distance to observation for NPE and ACE with $\beta = \{25, 50, 100\}$. Figure copied from Gao, Deistler & Macke [34].

108]. Previous results had inferred such parameters with a computationally expensive active learning scheme which was not amortized, therefore requiring to re-run inference for every datapoint [20]. We surmised that the failure of neural posterior estimation (NPE) to infer the posterior distribution was due to the simulator being, to some degree, misspecified. Based on our results on benchmark tasks, we expected that ACE would excel at this task. Indeed, we found that ACE could estimate the cost well (Fig. 3.9b), and that the predictive samples given the generalized posterior were significantly closer to the observation than samples from the NPE posterior (Fig. 3.9c,d). ACE was also significantly more simulation-efficient than previous attempts at inferring the posterior distribution of this simulator [20].

Finally, we studied the degeneracy of the generalized posterior. We found that, although ACE posterior predictives were much closer to the observation than NPE posterior predictives, the marginal distribution of the ACE posterior still exhibited a large amount of degeneracy (Fig. 3.9e). This suggests that the finding of degeneracy in biophysical, Hodgkin–Huxley-type models is not merely an artifact of the inference procedure (and the potential limitations of Bayesian inference), but that this finding is robust to inference targets.

Discussion

Bayesian inference is a powerful method to infer the space of data-compatible parameters, thereby allowing neuroscientists to study degeneracy. In cases where

the model is not an exact representation of reality, Bayesian inference can be overly restrictive: It excludes parameter sets that cannot *exactly* match the data and it is ill-defined for misspecified models. Here, we proposed to use generalized Bayesian inference to suitably adjust the inference target.

One of the benefits of generalized Bayesian inference is that it can be more robust to misspecified observations. Over the past years, misspecification has become a popular research topic in simulation-based Bayesian inference [67, 109–112]. Many of these papers extend the generative model—the simulator—with additional noise models to provide the simulator with more flexibility and to allow it to fit larger classes of observations. Here, we took a different approach: Instead of modifying the simulation, we directly modified the target of inference.

Overall, I believe that considering the sensibility of the inference target (or the generative model, including its noise model) is essential to reliable inference. I hope that our work on generalized Bayesian inference contributes to a robust, reliable, and responsible use of inference techniques for scientific models.

3.4 `sbi`: A TOOLKIT FOR SIMULATION-BASED BAYESIAN INFERENCE

The initial software paper "`sbi`: A toolkit for simulation-based inference" was published in the Journal of Open Source Software (JOSS) [35] and an earlier version of the software paper was made available as a preprint [113]. An updated version of the software paper, "`sbi` reloaded: A toolkit for simulation-based inference workflows", which reflects the state of the toolbox as of November 2024, is available as a preprint [36] and is currently under review at a journal. The `sbi` toolbox is openly available on Github: `https://github.com/sbi-dev/sbi`.

Statement of need

We realized early on that neural simulation-based Bayesian inference had great potential for applications to scientific simulators across many domains. However, re-implementing these methods could be challenging for domain scientists. In early 2020, we developed the `sbi` toolkit, which openly implements several methods for simulation-based Bayesian inference. The toolbox quickly gained many users, and we have been actively maintaining the toolbox for almost five years now.

Features of the `sbi` toolbox

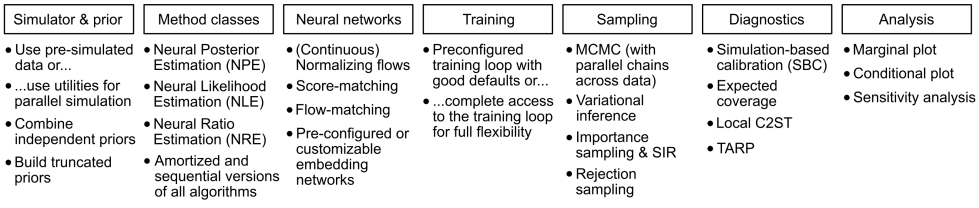


FIGURE 3.10: **Features of the `sbi` package.** The `sbi` package contains a wide range of algorithms, neural networks, sampling methods, and diagnostics. For every step of the inference process, it provides strong default values, but it also allows the user to have full flexibility if desired. Figure copied from Boelts and Deistler *et al.* [36].

The `sbi` toolbox implements three classes of neural simulation-based Bayesian inference methods: Neural Posterior Estimation (NPE), Neural Likelihood Estimation (NLE), and Neural Ratio Estimation (NRE). For all of these methods, `sbi` supports an *amortized* mode which allows inferring the posterior distribution for any observation $\boldsymbol{x}_o$, as well as a *sequential* mode which uses active learning to improve simulation efficiency. The `sbi` toolkit aims to provide strong default values for any step of the inference process, while also enabling full flexibility where desired.

Since its initial release, `sbi` has grown into one of the main toolkits for neural simulation-based Bayesian inference methods. Thanks to our continued efforts, as well as contributions by 63 scientists and engineers, the toolbox now implements

numerous methods, diagnostics, and analysis tools. To give a few examples: While early versions of `sbi` required passing a Python-compatible simulator, the current version of `sbi` provides full flexibility for offline simulations, and (optionally) also grants full flexibility over the training loop. While early versions had only a limited number of density estimators and no pre-configured embedding networks, the newest version has a large range of density estimators and pre-built embedding networks. While early versions could perform inference only for a single datapoint, `sbi` can now perform inference for arbitrary i.i.d.-sampled datapoints. While early versions of the toolbox only implemented MCMC sampling for NLE and NRE, the current version implements MCMC, variational inference [47], rejection sampling, and importance sampling. While the first version of `sbi` only implemented a limited set of neural networks, the `sbi` toolbox now also implements flow-matching and score-matching methods. While the first version implemented no diagnostics, it now implements simulation-based calibration [87, 88], expected coverage [33, 114], local C2ST [115], and TARP [116]. Overall, the `sbi` toolbox now allows scientists to run full Bayesian workflows for simulation-based models.

Discussion

The `sbi` toolbox has been used in dozens of applications in the sciences [19, 24, 25, 32, 38, 40, 69, 78–82, 117–123] and is used extensively by the machine learning research community [33, 34, 42, 45, 92, 115, 124–130]. I am particularly proud of the community that has grown around the toolbox. I hope to stay part of this community for years to come and I hope that the `sbi` toolbox will continue to enable domain scientists to apply neural SBI methods to gain insights into the mechanisms underlying experimental observations.

4

DIFFERENTIABLE SIMULATION FOR BIOPHYSICAL MODELS IN NEUROSCIENCE

Neural simulation-based Bayesian inference (SBI) methods are powerful tools for inferring parameters of simulators. However, they are typically limited to at most a few dozen parameters. In addition, SBI methods can struggle when the data is made up of several protocols, each corresponding to the response to a different input to the simulation. In that case, SBI methods like Neural Posterior Estimation (NPE) would require running the simulator as many times as there are protocols in order to generate a single training datapoint for the neural network. As NPE requires *at least* 1000 training datapoints to perform well [84], this becomes infeasible even for moderate numbers of protocols and moderately expensive simulators.

Contrary to these methods, gradient descent—the workhorse underlying the success of deep learning—excels at scaling to millions (or even billions) of parameters and at scaling to very large datasets of inputs and labels (such as stimuli and responses across many protocols). Despite this potential, at the time of starting my PhD, there was no biophysics simulator which allowed to compute gradients with automatic differentiation. This limited any investigation of how efficient and scalable gradient descent could be, how prone it is to local minima or overfitting, or what good loss functions or optimizers for biophysical models would look like [43].

To overcome these limitations, we built a differentiable biophysics simulator, Jaxley. Below, we describe our experiments that show the efficiency of Jaxley and its potential to train large-scale biophysical models with thousands of parameters. We then describe our openly available software toolbox which made these results possible and which allows domain scientists to infer parameters with gradient descent.

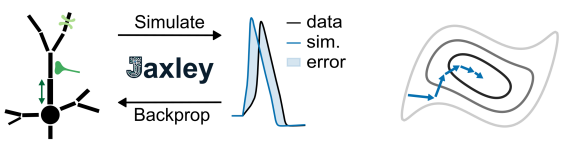


Figure 4.1: **Differentiable simulation enables gradient descent.** Left: We built Jaxley, a new simulator for differentiable neuroscience. Jaxley allows to accurately simulate biophysical models, and it uses automatic differentiation to compute the gradient with backpropagation of error. Right: By making gradients available, Jaxley opens up possibilities to optimize biophysical models with gradient descent on a loss function (contours).

4.1 DIFFERENTIABLE SIMULATION ENABLES LARGE-SCALE TRAINING OF DETAILED BIOPHYSICAL MODELS OF NEURAL DYNAMICS

Our paper "Differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics" is made available as a preprint [37] and is currently under peer-review at a journal.

JAXLEY: *A differentiable and GPU-compatible neuron simulator*

In order to study whether gradient descent could be used to improve parameter inference in biophysical models, we built a differentiable neuroscience simulator in JAX [132]. This simulator, which we named JAXLEY (JAX & [Hodgkin–Hux]ley)[1], can be used to simulate biophysically detailed neuron models and to compute the gradient with respect to parameters with automatic differentiation. For accurate simulation of such systems, JAXLEY clones the differential equation solver of the NEURON package into JAX[2]. Across a range of neuron morphologies and stimuli [131, 133], the solver of JAXLEY closely matched the voltage traces of the NEURON toolkit (Fig. 4.2a). However, thanks to being fully implemented in JAX, JAXLEY enables native GPU-parallelization [134–136], which can speed up simulation by more than

1 The name for the package came up in discussions with Auguste Schulz, Janne Lappalainen, and Richard Gao.

2 The exact solver is not spelled out in detail in a single resource. We heavily relied on the NEURON book and forum to reverse engineer the formula underlying the solver.

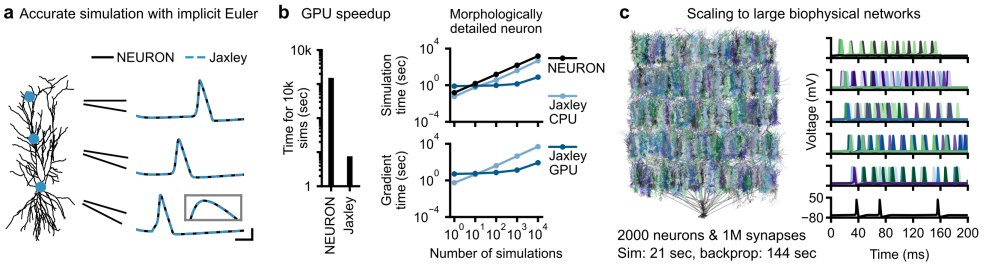


FIGURE 4.2: **Differentiable simulation enables training biophysical neuron models.** (a) Benchmarking JAXLEY. Simulated voltage traces at three locations, based on a reconstruction of a CA1 neuron [131] in response to a step current obtained with the NEURON simulator and with JAXLEY. Inset is a zoom-in to the peak of the action potential. Scalebars: 3 ms and 30 mV. (b) Left: Time to run 10k simulations with NEURON on CPU and with JAXLEY on GPU. Right: Simulation time (top) for the CA1 neuron shown in panel a and for a point neuron, as a function of number of simulations. Bottom: Same as top, for computing the gradient with backprop. (c) JAXLEY scales to large networks. JAXLEY can simulate and differentiate through a biophysically detailed network built from reconstructions of CA1 neurons (left, network consists of 2,000 neurons and 1M synapses) in response to step currents to the first layer (voltage responses to the right). Runtimes were evaluated on an A100 GPU. Figure adapted from Deistler *et al.* [37].

two orders of magnitude for large networks or parameter sweeps (Fig. 4.2b, left). In addition, thanks to the ability of JAX to just-in-time compile code, JAXLEY is as fast as NEURON on CPU, and can also speed up simulations of simpler point-neuron models (Fig. 4.2b, right). Finally, JAXLEY can parallelize across cells, branches, and compartments within a network, allowing to scale to networks of thousands of biophysically detailed neurons and millions of synapses (Fig. 4.2c). In order to compute the gradient with backpropagation in such large-scale models, JAXLEY implements multi-level checkpointing [137], thereby allowing JAXLEY to compute the gradient in simulations whose computation graph is multiple terabytes—on a single GPU.

JAXLEY *can fit biophysical single-neuron models to experimental recordings*

Our main drive to implement JAXLEY was to obtain gradients with respect to parameters with backpropagation of error [138]. We used this ability on a series of tasks, ranging from fitting single-neuron models to electrophysiology recordings to training networks of biophysically detailed neurons to solve computer vision tasks. We started with a set of single-neuron modeling tasks. First, we trained single neuron models to fit electrophysiology data [12, 139–141]. We trained 19 parameters of a single neuron model to match synthetic recordings and found that gradient descent is more simulation-efficient than a state-of-the-art genetic algorithm that was specifically designed to excel at this task (Fig. 4.3a,b,c). Taking into account the additional cost of computing the gradient with backpropagation, gradient descent had similar runtimes as the genetic algorithm on CPU. We then used this methodology to fit several simulations to match electrophysiology traces from the Allen Cell Types Database (Fig. 4.3d). Again, gradient descent found data-matching parameter configurations and had a similar runtime as a genetic algorithm. Overall, these results indicate that, even on models with relatively few parameters (and which can be efficiently optimized with gradient-free approaches), gradient descent is a viable alternative.

We then turned to models with significantly more parameters, expecting that the ability to evaluate gradients would be particularly useful in such models. We generated synthetic recordings from every branch of a layer 5 pyramidal cell (which could, in principle, be obtained experimentally with voltage imaging [142]) and fitted a single neuron model to these data. Unlike in the previous results, we fitted the conductance of every branch individually (and regularized the model to have similar conductance values in nearby branches [51, 143]), leading to more than 1,000 free parameters [144]. Despite this large parameter space, gradient descent quickly found data-matching solutions (Fig. 4.3g). In particular, while gradient descent solved this task within few gradient update steps, genetic algorithms failed to identify any solution with low loss (Fig. 4.3h). Overall, these results showed that gradient descent could be competitive with genetic algorithms on tasks on which these methods excel—and that gradient descent scales to simulations with many more parameters.

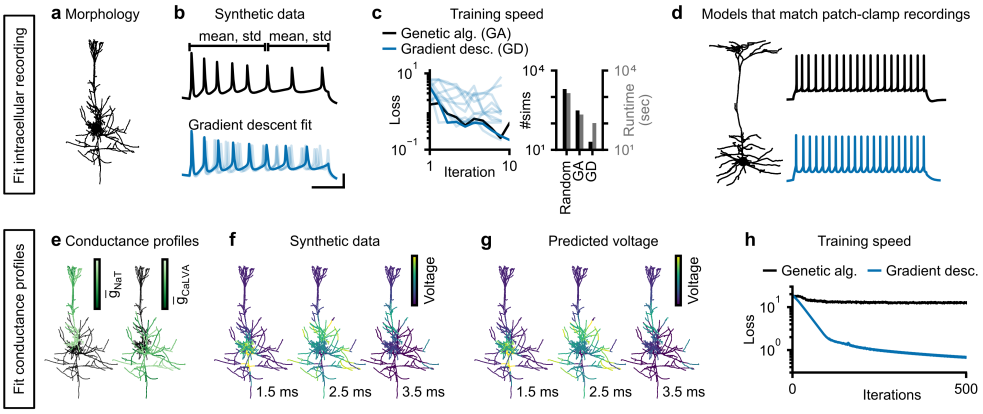


FIGURE 4.3: **Inferring single-neuron models with gradient descent. Task 1** (a) We optimize 19 parameters of a biophysical neuron model which is based on the morphology of a layer 5 pyramidal cell (L5PC) morphology. (b) Top: Synthetic somatic voltage recording (black) and windows that are used to compute summary statistics (top). Bottom: Fits obtained with gradient descent. Best fit in dark blue, fits from five best independent runs in light blue. Scalebars: 20 ms and 30 mV. (c) Left: Loss value of individual gradient descent runs (light blue), their minimum (dark blue), in comparison to the minimum loss across ten genetic algorithm runs (black). Right: Average number of simulations and runtime required to find a set of parameters with loss smaller than 0.55 with random sampling, genetic algorithm, and gradient descent. (d) We also fit morphologically detailed models to patch-clamp recordings (black) in response to step currents from the Allen Cell Types Database. Gradient descent fit in blue. Additional models in Deistler *et al.* [37]. Scalebars: 200 ms and 30 mV. **Task 2** (e) We optimize conductance profiles of the same L5PC morphology, leading to 1390 parameters. Synthetic ground-truth conductance profiles vary as a function of distance to the soma. (f) Simulated voltages given the synthetic conductance profile after 1.5, 2.5, and 3.5 ms. (g) Predicted voltages of gradient descent fit closely match synthetic observation. (h) Loss as a function of the number of iterations for gradient descent and genetic algorithm. Figure adapted from Deistler *et al.* [37].

JAXLEY *can fit network models to large-scale calcium recordings*

Having demonstrated the ability of JAXLEY to fit single-cell models, we next turned to network models. We developed a hybrid model of the calcium response of dendrites of retinal ganglion cells (RGCs). We modeled photoreceptors as linear filters, bipolar cells as single neurons with nonlinearities [145], and RGCs at full morphological detail with a variety of ion channels (Fig. 4.4a) [146]. We trained this model to predict dendritic calcium activity given a large dataset of random noise stimuli (Fig. 4.4b) [147]. After training, the simulated calcium response was positively correlated with the experimentally measured calcium activity on a held-out test set (Fig. 4.4c). In addition, the hybrid model exhibited compartmentalized responses to localized stimulation (Fig. 4.4d), a feature also inferred from recordings from RGCs [147]. We further investigated this behavior by computing the receptive fields of individual

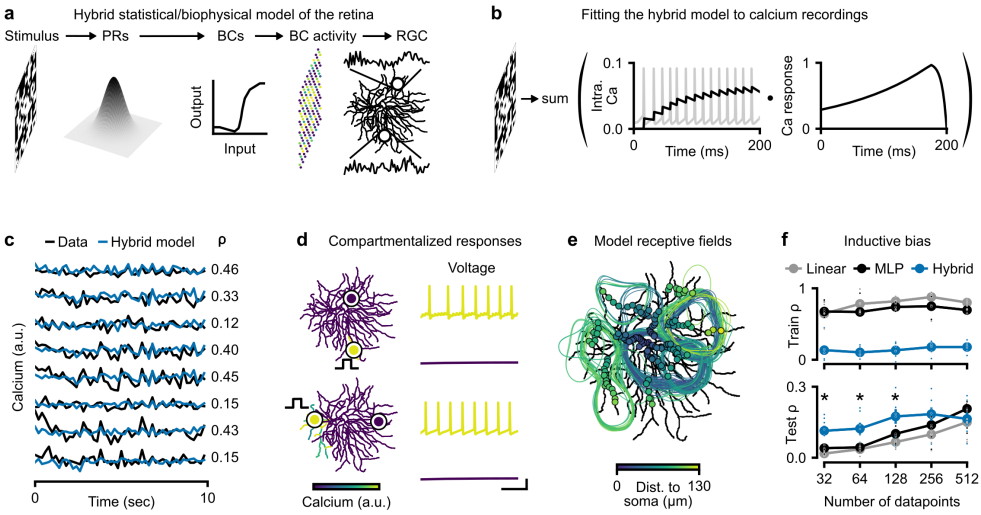


FIGURE 4.4: **Hybrid model of the calcium responses of a retinal ganglion cell.** (a) Schematic of experimental setup and hybrid model. (b) Schematic of training procedure and loss function. (c) Measured and model-predicted calcium response across 50 noise images (200 ms each). (d) Left: Calcium response (colormap) of the trained hybrid model to a step-current to a single branch indicated by step current sketch. Right: Voltage activity of the model at two branches, one at the stimulus site and one at a distant branch. Scalebars: 50 ms, 30 mV. (e) Receptive fields of the hybrid model obtained in response to 1024 noise stimuli. (f) Pearson correlation coefficient between experimental data and model for train (top) and test (bottom) data, for a linear network, a multi-layer perceptron, and the hybrid model. Error bars show standard-error of mean over seven datasets (see Methods). Asterisk denotes statistically significant difference between mean correlations of hybrid model and MLP (one-sided t-test at $p<0.05$). Figure adapted from Deistler *et al.* [37].

branches and found that, matching experimental recordings, they did not cover the entire cell (Fig. 4.4e). Finally, we evaluated whether the hybrid model could provide an inductive bias that avoids overfitting. We trained reduced models on a smaller dataset and compared the performance of our hybrid model to a multi-layer perceptron (MLP) trained on the same datasets. While the MLP outperformed the hybrid model on large datasets, the hybrid model indeed significantly outperformed the MLP on a held-out test set for small datasets (Fig. 4.4f). This demonstrates that biophysical models provide an inductive bias which could be used, for example, as a regularizer to black-box neural network models [148–150].

JAXLEY *can train biophysical neuron models on computational tasks*

The tasks above demonstrate that JAXLEY is a powerful tool to fit biophysical models to experimental recordings of voltage or calcium. We investigated whether the ability of JAXLEY to compute gradients would also allow it to train biophysical models to

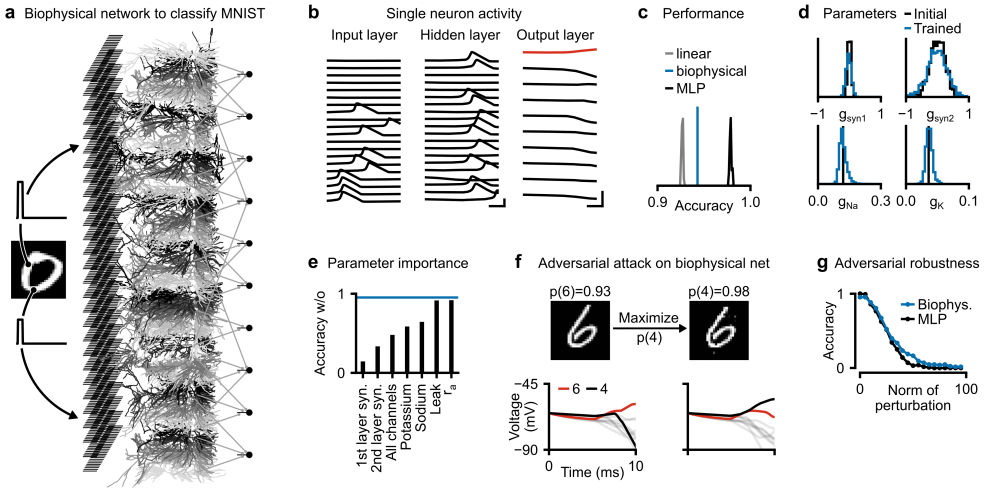


FIGURE 4.5: **Training biophysically detailed networks to solve computer vision tasks.** (a) We trained a biophysical network consisting of 28×28 input neurons, 64 morphologically detailed hidden neurons, and ten output neurons. (b) Voltages measured at the somata of neurons in the trained network in response to an image labelled as '0'. Red color in the output layer indicates the image label. Scalebars: 2 ms and 80 mV. (c) Histograms of test set accuracy of 50 linear networks (gray), 50 multi-layer perceptrons with 64 hidden neurons and ReLU activations (black), and the biophysical network (blue). (d) Histogram of parameters before (black) and after (blue) training. Training the network only leads to subtle shifts in parameter distributions. (e) Test set accuracy for trained network when subsets of parameters are reset to their initial value. Blue line is the fully trained network. (f) Adversarial attack on the biophysical network. (g) Accuracy across 128 test set examples, as a function of the norm of the adversarial image perturbation. Figure adapted from Deistler *et al.* [37].

solve *computational tasks* such as pattern recognition or computer vision tasks. To this end, we performed two additional experiments.

First, we trained recurrent neural networks (RNNs) to solve working memory tasks. Unlike typical RNNs, the neurons in our RNNs contained biophysically detailed mechanisms following Hodgkin–Huxley dynamics. Despite these biophysical constraints on the model, the resulting network could solve an evidence accumulation and a delayed-match-to-sample task [37].

Second, we trained a biophysical neural network to solve the MNIST computer vision task [136]. We built a network consisting of 28^2 input neurons, 64 hidden layer neurons modeled at full morphological detail, and 10 output neurons (Fig. 4.5a). The model had a total of 100k parameters (50k cellular and 50k synaptic). We trained this model with cross-entropy loss on the voltages of the output neurons (at the last time step, Fig. 4.5b). After training, the network could classify MNIST digits with an accuracy of 94.3%. While this is less than a multi-layer perceptron, the biophysical network outperforms an optimal linear classifier—despite not having

any nonlinearities that are not biologically plausible (Fig. 4.5c). This indicates that the biophysical network could leverage or adapt its existing nonlinearities to improve task performance beyond a linear classifier.

We then studied the trained model. Our first observation was that the distribution of parameters of the trained model was similar to the distribution at which the parameters were initialized (Fig. 4.5d). When resetting subsets of parameters to their initial value, however, performance dropped largely—for some sets of parameters even to chance level (Fig. 4.5e). This indicates that the relative tuning of parameters with respect to each other impacts task performance. As such, approaches which build biophysical networks based only on aggregate statistics of measured parameter values might fail to generate functional network activity.

In addition to fitting large-scale networks with gradient descent, the ability to compute gradients also opens up new possibilities for network analyses. To demonstrate this, we used gradients to perform adversarial attacks on our biophysical network [136]. We found that the biophysical network was, indeed, vulnerable to adversarial attacks (Fig. 4.5f), and did not show higher robustness than a multi-layer-perceptron with the same number of neurons (Fig. 4.5g).

Overall, these results demonstrate that gradient descent allows fitting large-scale biophysical models to perform computations. Therein, gradient descent can fit thousands of parameters of neural systems and enables new analyses for biophysical models.

Discussion

Fitting parameters of biophysical models is notoriously difficult [139]. While gradient-free approaches such as random search or genetic algorithms can perform well for models with few parameters, they typically struggle when models have hundreds, thousands, or millions of parameters. Inspired by the capabilities of gradient descent to optimize artificial neural networks, we developed a fully differentiable biophysics simulator. We then demonstrated that gradient descent can fit biophysical models to match voltage or calcium recordings, and it can train such models to solve computational tasks. We hope that our work enables new opportunities for fitting biophysical neuroscience simulations and that this will provide new insights into the biophysical mechanisms underlying biological intelligence.

4.2 THE JAXLEY SIMULATOR: DIFFERENTIABLE BIOPHYSICAL NEURON MODELS

The JAXLEY toolbox is openly available on Github: `https://github.com/jaxleyverse/jaxley`.

JAXLEY *is a user-friendly and flexible biophysics simulator*

Above, we showed that gradient descent is a highly efficient method for optimizing biophysical models of neural dynamics. However, enabling the neuroscience community to use these methods requires a flexible and user-friendly simulator that supports automatic differentiation—previous simulators such as NEURON or Brian2 do not support this [151, 152]. Because of this, we decided to build JAXLEY as a user-friendly and general-purpose toolbox for biophysical simulation. After the initial publication of JAXLEY, we have added many features and made JAXLEY more flexible and easy to interact with. Below, we outline the main features of JAXLEY.

Biophysical models come in a large variety, which poses a serious challenge for a simulation toolbox. For example, the toolbox must be able to deal with single neurons and with networks thereof, and every cell in a network must be fully customizable. An individual cell could have a particular channel in its basal dendrite, another channel in its apical dendrite, and both of these channels in its soma. On top of this, the channel models themselves can be extremely complex and must allow the user to, for example, share parameters across channels (e.g., different potassium channels all have the same reversal potential), or they must be able to depend on one another (e.g., a potassium channel could depend on the intracellular calcium concentration, which is modified by a calcium channel). Finally, computational neuroscientists should be able to flexibly insert stimuli into any part of a cell, and record voltage from another part.

JAXLEY provides flexible and elegant mechanisms to deal with this complexity of biophysical simulation. It allows users to implement any kind of channel model and enables them to insert these channel models into any compartment, branch, or cell in a network. JAXLEY also provides elegant mechanisms for interacting with networks or cells to, for example, modify parameters. These features allow JAXLEY to simulate a wide range of biophysical models, ranging from networks of passive (or rate-based) point neurons, via single-cell morphologically detailed models, to hybrid models of networks in which some cells are simulated at full morphological detail and others are largely simplified.

Elegant mechanisms for differentiation and parallelization

JAXLEY also implements elegant mechanisms for differentiation and parallelization. JAXLEY can compute the gradient with respect to any parameter in the system and can flexibly share parameters (to, e.g., have the same maximal sodium conductance

for all dendritic branches). JAXLEY can also compute the gradient with respect to inputs (e.g., for adversarial examples) or environmental conditions. On top of this, JAXLEY can perform forward- and backward-mode automatic differentiation and can compute higher order derivatives, thereby enabling easy implementations of advanced optimizers [153].

JAXLEY is a highly efficient simulator. It achieves this efficiency with parallelization and just-in-time compilation. JAXLEY automatically runs its solver in a way that optimally parallelizes computation. For example, the differential equations of the channels (where each compartment is independent of one another) are computed in parallel. In addition, JAXLEY allows the user to elegantly parallelize across parameter sets (e.g., for parameter sweeps or genetic algorithms) or stimuli (e.g., for multiple protocols). For all of these, JAXLEY can parallelize across multiple GPUs, enabling very large simulations or sweeps.

Discussion

With the JAXLEY toolkit, we aim to bring the power of gradient descent to computational neuroscientists. Motivated by our results which demonstrated that gradient descent could largely improve parameter estimation in biophysical models, we aimed to build a simulator that allows neuroscientists to use these tools. We built JAXLEY such that it is flexible, allowing it to simulate diverse biophysical models, while also enabling intuitive and easy interaction with the model. We hope that JAXLEY will become the go-to tool for biophysical simulation.

5

DISCUSSION AND FUTURE DIRECTIONS

Parameter inference is a central challenge in neuroscience: It is almost never possible to measure all microscopic and macroscopic properties of large neural systems. In addition, even minor measurement errors could, especially if accumulated across many cells or processes, lead to major errors in the resulting activity prediction. It is therefore necessary to *infer* from empirical data properties that cannot be measured.

In this thesis, we explored how current limitations in simulation-based parameter inference can be overcome by employing machine learning methods. We advanced two sets of methods for this: Neural simulation-based Bayesian inference (SBI) and differentiable simulation. In Chapter 3, we developed new SBI methods which are more scalable, applicable and robust than previous methods, and we presented a toolbox that allows domain scientists to apply these methods. In addition, we suggested that SBI can be a promising tool to study degeneracy in neuroscience and we studied the impact of metabolic cost on circuit degeneracy. In Chapter 4, we developed the first differentiable simulator for biophysical neuron models. Using this simulator, we trained large-scale biophysical neuron models with gradient descent, overcoming limitations on the number of parameters that can be inferred.

5.1 SUMMARY: SBI AND DIFFERENTIABLE SIMULATION

SBI methods have many advantages over previous inference methods [16]. They can perform inference for any (stochastic) black-box simulator, making them applicable to a broad range of domains, and they return the full posterior distribution, allowing scientists to study parameter uncertainty and degeneracy. Over the past years, SBI has become a popular method for parameter estimation in domains ranging from astrophysics [23–25] to economics [69] and neuroscience [78–81].

The generality of SBI, however, comes at a price: Almost any successful application of these methods infers only a limited number of parameters. Indeed, most applications infer fewer than ten parameters, and almost no application over the past years successfully inferred more than 100 parameters. This is because SBI methods, such as Neural Posterior Estimation (NPE), randomly sample parameters and then use a neural network to interpolate between the parameters. For simulators which require fine tuning of a large set of parameters, any kind of interpolation can become insufficient. Instead, one requires methods which actively search for optimal parameter sets. While active learning methods can largely improve simulation-efficiency, they typically do not enable scaling to thousands of parameters [33, 84].

In contrast, gradient descent can identify millions or billions of free parameters of artificial neural networks. By implementing mechanistic models in a differentiable programming language, *differentiable simulation* aims to bring the potential of fitting

many parameters to mechanistic modeling. As we demonstrated, differentiable simulation enables fitting large-scale biophysical models with an unprecedented number of parameters.

To further improve inference, SBI and gradient descent could even be efficiently combined: As a first step, SBI can be used to infer the posterior distribution of single-cell parameters given single-cell electrophysiological recordings $\boldsymbol{x}_{\text{data}}$. This returns a fully differentiable posterior $q(\boldsymbol{\theta}|\boldsymbol{x}_{\text{data}})$ that constrains cellular parameters by measurements, but it does not yet fix parameters to a single set of "best" parameters:

$$q(\boldsymbol{\theta}|\boldsymbol{x}_{\text{data}}) \approx \frac{p(\boldsymbol{x}_{\text{data}}|\boldsymbol{\theta}) \cdot p(\boldsymbol{\theta})}{p(\boldsymbol{x}_{\text{data}})}.$$

As a second step, individual cells can then be assembled into a network whose parameters can be optimized with gradient descent to perform a computational task $\boldsymbol{x}_{\text{task}}$—using the approximate posterior of individual cells as regularizer. This returns the maximum-a-posteriori estimate given data- and task-constraints (under the assumption that data and task are conditionally independent):

$$\begin{aligned}\arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{x}_{\text{data}}, \boldsymbol{x}_{\text{task}}) &= \arg\max_{\boldsymbol{\theta}} \log\left(p(\boldsymbol{x}_{\text{task}}|\boldsymbol{\theta}) \cdot p(\boldsymbol{x}_{\text{data}}|\boldsymbol{\theta}) \cdot p(\boldsymbol{\theta})\right) \\ &\approx \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x}_{\text{task}}|\boldsymbol{\theta}) + \log q(\boldsymbol{\theta}|\boldsymbol{x}_{\text{data}}).\end{aligned}$$

Such workflows could enable building large data- and task-constrained models of neural dynamics.

5.2 IS BIOPHYSICALLY DETAILED NEURON MODELING REALISTIC?

Throughout this thesis, we aimed to infer properties of neural systems from activity measurements. We motivated this process by the assumption that some properties of a neural system—those we do *not* aim to infer—can be measured, and our model would contain the true value for these properties. In neuroscience, more often than not, this is not the case. The existence of channels in particular parts of a cell is often merely hypothesized, or is based on findings of said channel in different cells, often in different animals. On top of this, many processes that are known to occur in cells or networks are neglected for computational purposes. Finally, the connectivity of cells in networks is rarely measured exactly, but is often assumed to follow aggregate statistics of network parameters.

Given such inaccuracies of the model, multiple concerns for parameter inference arise. Firstly, with these limitations in mind, one has to ask: Even if we did find the optimal parameters with simulation-based parameter inference, would the model make any meaningful predictions? It has been argued that, even for the most realistic single neuron models, this is not the case [10]. To alleviate these concerns, it is necessary to validate predictions of computational models with experimental recordings.

Secondly, inference is typically motivated by obtaining as good as possible a value for properties that cannot be measured. If the model is largely misspecified,

however, it is unlikely that any inference method would return an optimal estimate for said property. Instead, for such models, inference methods should rather be considered as parameter *fitting* methods which make a model match experimental recordings, but should not be—without further validation—be trusted to return reasonable parameter estimates. Despite this limitation, parameter fitting can still be useful: It allows scientists to study mechanisms in a fitted model, which can lead to insight and hypotheses for better experiments.

5.3 ON IDENTIFIABILITY FOR PARAMETER ESTIMATION IN NEUROSCIENCE

We have argued that the number of free parameters is a fundamental challenge for building biophysical models. Indeed, big parts of this thesis aim to fit more detailed models with an increasing number of parameters. While this approach can be promising, it also bears danger: Being able to fit millions of parameters to, for example, a single recording, does not mean that the identified parameters will be the true parameters. Indeed, by identifying a single best-fitting parameter set given a limited number of recordings, one is likely to *overfit* and obtain poor predictions on unseen stimuli or protocols.

The most pragmatic way to alleviate these concerns is to collect more data—or more detailed data. For example, somatic recordings might not constrain parameters in the apical dendrite, but, if one had voltage recordings from every branch of the cell, at high spatial and temporal resolution, it might be possible to more strongly constrain parameters. Such data are becoming increasingly available with experimental techniques such as voltage imaging [154]. Another possibility to alleviate overfitting is to modify the inference method. By providing a Bayesian estimate of parameters, SBI methods return a range of plausible models, each consistent with the observations. For simulators that are out of reach for SBI (e.g., due to too many parameters), gradient-based Bayesian inference methods such as Hamiltonian Monte–Carlo [155], ensembles [156], or Laplace approximations [157] could provide an alternative.

5.4 TOWARDS A NEW GENERATION OF MODELS IN NEUROSCIENCE

Computational simulators are typically run in the forward direction: They generate predictions given a particular set of parameters. In many cases, however, the inverse direction can be equally relevant: Given observed data, what would be the parameters [158]? By answering this question one can recover properties which would otherwise not be measurable and one can construct computer simulations that are consistent with observations. Most simulators in science, however, are not designed with the inverse direction in mind: Computer simulators can be stochastic, can be slow to simulate, and can have many parameters, making it particularly challenging to perform inference.

This thesis presented machine learning methods that allow scientists to infer parameters given mechanistic simulators and experimental measurements. By making inference possible for a wide range of models, we enabled new kinds of analyses and a targeted design of experiments. We hope that this thesis contributes to a responsible yet impactful use of machine learning methods for scientific discovery and that the tools presented here are a stepping stone for future research on parameter inference in neuroscience and beyond.

REFERENCES

1. Hodgkin, A. L. & Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology* **117**, 500 (1952).
2. Rall, W. Electrophysiology of a dendritic neuron model. *Biophysical journal* **2**, 145 (1962).
3. Hines, M. L. & Carnevale, N. T. The NEURON simulation environment. *Neural computation* **9**, 1179 (1997).
4. Bower, J. M. & Beeman, D. *The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System* (Springer Science & Business Media, 2012).
5. De Schutter, E. Computer software for development and simulation of compartmental models of neurons. *Computers in biology and medicine* **19**, 71 (1989).
6. Hodgkin, A. L. & Huxley, A. F. The components of membrane conductance in the giant axon of Loligo. *The Journal of physiology* **116**, 473 (1952).
7. Hodgkin, A. L. & Huxley, A. F. The dual effect of membrane potential on sodium conductance in the giant axon of Loligo. *The Journal of physiology* **116**, 497 (1952).
8. Hodgkin, A. L. & Huxley, A. F. Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo. *The Journal of physiology* **116**, 449 (1952).
9. Hodgkin, A. L., Huxley, A. F. & Katz, B. Measurement of current-voltage relations in the membrane of the giant axon of Loligo. *The Journal of physiology* **116**, 424 (1952).
10. Almog, M. & Korngreen, A. Is realistic neuronal modeling realistic? *Journal of neurophysiology* **116**, 2180 (2016).
11. Hines, M. Efficient computation of branched nerve equations. *International journal of bio-medical computing* **15**, 69 (1984).
12. Markram, H., Muller, E., Ramaswamy, S., Reimann, M. W., Abdellah, M., Sanchez, C. A., Ailamaki, A., Alonso-Nanclares, L., Antille, N., Arsever, S., *et al.* Reconstruction and simulation of neocortical microcircuitry. *Cell* **163**, 456 (2015).
13. Prinz, A. A., Bucher, D. & Marder, E. Similar network activity from disparate circuit parameters. *Nature neuroscience* **7**, 1345 (2004).
14. Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* **25** (2012).

15. Lavin, A., Krakauer, D., Zenil, H., Gottschlich, J., Mattson, T., Brehmer, J., Anandkumar, A., Choudry, S., Rocki, K., Baydin, A. G., *et al.* Simulation intelligence: Towards a new generation of scientific methods. *arXiv preprint arXiv:2112.03235* (2021).

16. Cranmer, K., Brehmer, J. & Louppe, G. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences* **117**, 30055 (2020).

17. Cranmer, K. & Lo, J. *Simulation-Based Inference Blog* `http://simulation-based-inference.org`. Accessed: 2025-02-28. 2024.

18. Brehmer, J. & Cranmer, K. in *Artificial Intelligence for High Energy Physics* 579 (World Scientific, 2022).

19. Wang, X., Kelly, R. P., Jenner, A. L., Warne, D. J. & Drovandi, C. A Comprehensive Guide to Simulation-based Inference in Computational Biology. *arXiv preprint arXiv:2409.19675* (2024).

20. 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., *et al.* Training deep neural density estimators to identify mechanistic models of neural dynamics. *Elife* **9**, e56261 (2020).

21. Karimi, H. S., Pal, A., Ning, L. & Rathi, Y. Likelihood-free posterior estimation and uncertainty quantification for diffusion MRI models. *Imaging Neuroscience* **2**, 1 (2024).

22. Manzano-Patrón, J., Deistler, M., Schröder, C., Kypraios, T., Gonçalves, P. J., Macke, J. H. & Sotiropoulos, S. S. Uncertainty mapping and probabilistic tractography using Simulation-Based Inference in diffusion MRI: A comparison with classical Bayes. *bioRxiv*, 2024 (2024).

23. Dax, M., Green, S. R., Gair, J., Macke, J. H., Buonanno, A. & Schölkopf, B. Real-time gravitational wave science with neural posterior estimation. *Physical review letters* **127**, 241103 (2021).

24. Mishra-Sharma, S. & Cranmer, K. Neural simulation-based inference approach for characterizing the Galactic Center γ-ray excess. *Physical Review D* **105**, 063017 (2022).

25. Lemos, P., Parker, L., Hahn, C., Ho, S., Eickenberg, M., Hou, J., Massara, E., Modi, C., Dizgah, A. M., Blancard, B. R.-S., *et al.* Field-level simulation-based inference of galaxy clustering with convolutional neural networks. *Physical Review D* **109**, 083536 (2024).

26. Baydin, A. G., Pearlmutter, B. A., Radul, A. A. & Siskind, J. M. Automatic differentiation in machine learning: a survey. *Journal of machine learning research* **18**, 1 (2018).

27. Schoenholz, S. & Cubuk, E. D. Jax md: a framework for differentiable physics. *Advances in Neural Information Processing Systems* **33**, 11428 (2020).

28. Holl, P., Thuerey, N. & Koltun, V. *Learning to Control PDEs with Differentiable Physics* in *International Conference on Learning Representations* (2020).

29. AlQuraishi, M. & Sorger, P. K. Differentiable biology: using deep learning for biophysics-based and data-driven modeling of molecular mechanisms. *Nature methods* **18**, 1169 (2021).

30. Dorigo, T., Giammanco, A., Vischia, P., Aehle, M., Bawaj, M., Boldyrev, A., de Castro Manzano, P., Derkach, D., Donini, J., Edelen, A., *et al.* Toward the end-to-end optimization of particle physics instruments with differentiable programming. *Reviews in Physics*, 100085 (2023).

31. Shen, C., Appling, A. P., Gentine, P., Bandai, T., Gupta, H., Tartakovsky, A., Baity-Jesi, M., Fenicia, F., Kifer, D., Li, L., *et al.* Differentiable modelling to unify machine learning and physical models for geosciences. *Nature Reviews Earth & Environment* **4**, 552 (2023).

32. Deistler, M., Macke, J. H. & Gonçalves, P. J. Energy-efficient network activity from disparate circuit parameters. *Proceedings of the National Academy of Sciences* **119**, e2207632119 (2022).

33. Deistler, M., Goncalves, P. J. & Macke, J. H. Truncated proposals for scalable and hassle-free simulation-based inference. *Advances in Neural Information Processing Systems* **35**, 23135 (2022).

34. Gao, R., Deistler, M. & Macke, J. H. Generalized Bayesian inference for scientific simulators via amortized cost estimation. *Advances in Neural Information Processing Systems* **36**, 80191 (2023).

35. Tejero-Cantero, A., Boelts, J., Deistler, M., Lueckmann, J.-M., Durkan, C., Gonçalves, P. J., Greenberg, D. S. & Macke, J. H. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software* **5**, 2505 (2020).

36. Boelts, J., Deistler, M., Gloeckler, M., Tejero-Cantero, Á., Lueckmann, J.-M., Moss, G., Steinbach, P., Moreau, T., Muratore, F., Linhart, J., *et al.* sbi reloaded: a toolkit for simulation-based inference workflows. *arXiv preprint arXiv:2411.17337* (2024).

37. Deistler, M., Kadhim, K. L., Pals, M., Beck, J., Huang, Z., Gloeckler, M., Lappalainen, J. K., Schröder, C., Berens, P., Gonçalves, P. J., *et al.* Differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics. *bioRxiv*, 2024 (2024).

38. Gao, R., Deistler, M., Schulz, A., Gonçalves, P. J. & Macke, J. H. Deep inverse modeling reveals dynamic-dependent invariances in neural circuit mechanisms. *bioRxiv*, 2024 (2024).

39. Motallebzadeh, H., Deistler, M., Schönleitner, F. M., Macke, J. H. & Puria, S. From Simulations to Inference: Using Machine Learning to Tune Patient-Specific Finite-Element Models of the Middle Ear Towards Objective Diagnosis. *bioRxiv* (2024).

40. Bernaerts, Y., Deistler, M., Gonçalves, P. J., Beck, J., Stimberg, M., Scala, F., Tolias, A. S., Macke, J., Kobak, D. & Berens, P. Combined statistical-mechanistic modeling links ion channel genes to physiology of cortical neuron types. *bioRxiv*, 2023 (2023).
41. Bischoff, S., Darcher, A., Deistler, M., Gao, R., Gerken, F., Gloeckler, M., Haxel, L., Kapoor, J., Lappalainen, J. K., Macke, J. H., Moss, G., Pals, M., Pei, F. C., Rapp, R., Sağtekin, A. E., Schröder, C., Schulz, A., Stefanidi, Z., Toyota, S., Ulmer, L. & Vetter, J. A Practical Guide to Sample-based Statistical Distances for Evaluating Generative Models in Science. *Transactions on Machine Learning Research* (2024).
42. Gloeckler, M., Deistler, M., Weilbach, C. D., Wood, F. & Macke, J. H. *All-in-one simulation-based inference* in *Forty-first International Conference on Machine Learning* (2024).
43. Beck, J., Bosch, N., Deistler, M., Kadhim, K. L., Macke, J. H., Hennig, P. & Berens, P. *Diffusion Tempering Improves Parameter Estimation with Probabilistic Integrators for Ordinary Differential Equations* in *Forty-first International Conference on Machine Learning* (2024).
44. Gorecki, M., Macke, J. H. & Deistler, M. Amortized Bayesian Decision Making for simulation-based models. *Transactions on Machine Learning Research* (2024).
45. Gloeckler, M., Deistler, M. & Macke, J. H. *Adversarial robustness of amortized Bayesian inference* in *International Conference on Machine Learning* (2023), 11493.
46. Beck, J., Deistler, M., Bernaerts, Y., Macke, J. H. & Berens, P. Efficient identification of informative features in simulation-based inference. *Advances in Neural Information Processing Systems* **35**, 19260 (2022).
47. Glöckler, M., Deistler, M. & Macke, J. H. *Variational methods for simulation-based inference* in *International Conference on Learning Representations* (2022).
48. Dax, M., Green, S. R., Gair, J., Deistler, M., Schölkopf, B. & Macke, J. H. *Group equivariant neural posterior estimation* in *International Conference on Learning Representations* (2022).
49. Stone, H. S. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the ACM (JACM)* **20**, 27 (1973).
50. Carnevale, N. T. & Hines, M. L. *The NEURON book* (Cambridge University Press, 2006).
51. Migliore, M. & Shepherd, G. M. Emerging rules for the distributions of active dendritic conductances. *Nature Reviews Neuroscience* **3**, 362 (2002).
52. Goldwyn, J. H. & Shea-Brown, E. The what and where of adding channel noise to the Hodgkin-Huxley equations. *PLoS computational biology* **7**, e1002247 (2011).
53. Hennig, P., Osborne, M. A. & Girolami, M. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **471**, 20150142 (2015).

54. Prinz, A. A., Billimoria, C. P. & Marder, E. Alternative to Hand-Tuning Conductance-Based Models: Construction and Analysis of Databases of Model Neurons. *Journal of Neurophysiology* **90**, 3998 (2003).

55. Edelman, G. M. & Gally, J. A. Degeneracy and complexity in biological systems. *Proceedings of the national academy of sciences* **98**, 13763 (2001).

56. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. Equation of state calculations by fast computing machines. *The journal of chemical physics* **21**, 1087 (1953).

57. Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American statistical Association* **112**, 859 (2017).

58. Papamakarios, G. & Murray, I. Fast ε-free inference of simulation models with bayesian conditional density estimation. *Advances in neural information processing systems* **29** (2016).

59. Papamakarios, G., Sterratt, D. & Murray, I. *Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows* in *The 22nd International Conference on Artificial Intelligence and Statistics* (2019), 837.

60. Cranmer, K., Pavez, J. & Louppe, G. Approximating likelihood ratios with calibrated discriminative classifiers. *arXiv preprint arXiv:1506.02169* (2015).

61. Hermans, J., Begy, V. & Louppe, G. *Likelihood-free mcmc with amortized approximate ratio estimators* in *International Conference on Machine Learning* (2020), 4239.

62. Lueckmann, J.-M., Goncalves, P. J., Bassetto, G., Öcal, K., Nonnenmacher, M. & Macke, J. H. Flexible statistical inference for mechanistic models of neural dynamics. *Advances in neural information processing systems* **30** (2017).

63. Greenberg, D., Nonnenmacher, M. & Macke, J. *Automatic posterior transformation for likelihood-free inference* in *International Conference on Machine Learning* (2019), 2404.

64. Brehmer, J., Louppe, G., Pavez, J. & Cranmer, K. Mining gold from implicit models to improve likelihood-free inference. *Proceedings of the National Academy of Sciences* **117**, 5242 (2020).

65. Baur, W. & Strassen, V. The complexity of partial derivatives. *Theoretical computer science* **22**, 317 (1983).

66. Golowasch, J., Goldman, M. S., Abbott, L. & Marder, E. Failure of averaging in the construction of a conductance-based neuron model. *Journal of neurophysiology* **87**, 1129 (2002).

67. Cannon, P., Ward, D. & Schmon, S. M. Investigating the impact of model misspecification in neural simulation-based inference. *arXiv preprint arXiv:2209.01845* (2022).

68. Korfmann, K., Gaggiotti, O. E. & Fumagalli, M. Deep learning in population genetics. *Genome Biology and Evolution* **15**, evad008 (2023).

69. Dyer, J., Cannon, P., Farmer, J. D. & Schmon, S. Black-box Bayesian inference for economic agent-based models. *arXiv preprint arXiv:2202.00625* (2022).
70. Deistler, M., Macke, J. H. & Gonçalves, P. J. Energy efficient network activity from disparate circuit parameters. *bioRxiv*, 2021 (2021).
71. Marder, E. & Bucher, D. Understanding circuit dynamics using the stomatogastric nervous system of lobsters and crabs. *Annu. Rev. Physiol.* **69**, 291 (2007).
72. Moujahid, A., d'Anjou, A., Torrealdea, F. & Torrealdea, F. Energy and information in Hodgkin-Huxley neurons. *Physical Review E* **83**, 031912 (2011).
73. Haddad, S. A. & Marder, E. *Recordings from the C. borealis Stomatogastric Nervous System at different temperatures in the decentralized condition* 2021.
74. Stehlik, L., MacKenzie Jr, C. & Morse, W. Distribution and abundance of four brachyuran crabs on the northwest Atlantic shelf. *Fishery Bulletin* **89**, 473 (1991).
75. Donahue, M. J., Nichols, A., Santamaria, C. A., League-Pike, P. E., Krediet, C. J., Perez, K. O. & Shulman, M. J. Predation risk, prey abundance, and the vertical distribution of three brachyuran crabs on Gulf of Maine shores. *Journal of Crustacean Biology* **29**, 523 (2009).
76. Krediet, C. J. & Donahue, M. J. Growth-mortality trade-offs along a depth gradient in Cancer borealis. *Journal of Experimental Marine Biology and Ecology* **373**, 133 (2009).
77. Haddad, S. A. & Marder, E. Circuit robustness to temperature perturbation is altered by neuromodulators. *Neuron* **100**, 609 (2018).
78. Rößler, N., Jungenitz, T., Sigler, A., Bird, A., Mittag, M., Rhee, J. S., Deller, T., Cuntz, H., Brose, N., Schwarzacher, S. W., *et al.* Skewed distribution of spines is independent of presynaptic transmitter release and synaptic plasticity, and emerges early during adult neurogenesis. *Open Biology* **13**, 230063 (2023).
79. Confavreux, B., Ramesh, P., Goncalves, P. J., Macke, J. H. & Vogels, T. Meta-learning families of plasticity rules in recurrent spiking networks using simulation-based inference. *Advances in Neural Information Processing Systems* **36**, 13545 (2023).
80. Groschner, L. N., Malis, J. G., Zuidinga, B. & Borst, A. A biophysical account of multiplication by a single neuron. *Nature* **603**, 119 (2022).
81. Myers-Joseph, D., Wilmes, K. A., Fernandez-Otero, M., Clopath, C. & Khan, A. G. Disinhibition by VIP interneurons is orthogonal to cross-modal attentional modulation in primary visual cortex. *Neuron* **112**, 628 (2024).
82. Jin, H., Verma, P., Jiang, F., Nagarajan, S. S. & Raj, A. Bayesian inference of a spectral graph model for brain oscillations. *NeuroImage* **279**, 120278 (2023).
83. Deistler, M., Goncalves, P. J. & Macke, J. H. Truncated proposals for scalable and hassle-free simulation-based inference. *arXiv preprint arXiv:2210.04815* (2022).

84. Lueckmann, J.-M., Boelts, J., Greenberg, D., Goncalves, P. & Macke, J. *Benchmarking simulation-based inference* in *International Conference on Artificial Intelligence and Statistics* (2021), 343.

85. Blum, M. G. & François, O. Non-linear regression models for Approximate Bayesian Computation. *Statistics and Computing* **20**, 63 (2010).

86. Miller, B., Cole, A., Forré, P., Louppe, G. & Weniger, C. Truncated marginal neural ratio estimation. *Advances in Neural Information Processing Systems* **34** (2021).

87. Cook, S. R., Gelman, A. & Rubin, D. B. Validation of software for Bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics* **15**, 675 (2006).

88. Talts, S., Betancourt, M., Simpson, D., Vehtari, A. & Gelman, A. Validating Bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:1804.06788* (2018).

89. Dalmasso, N., Pospisil, T., Lee, A. B., Izbicki, R., Freeman, P. E. & Malz, A. I. Conditional density estimation tools in python and R with applications to photometric redshifts and likelihood-free cosmological inference. *Astronomy and Computing* **30**, 100362 (2020).

90. Hermans, J., Delaunoy, A., Rozet, F., Wehenkel, A. & Louppe, G. Averting a crisis in simulation-based inference. *arXiv preprint arXiv:2110.06581* (2021).

91. Rozet, F. *et al.* Arbitrary marginal neural ratio estimation for likelihood-free inference. *Université de Liège, Liège, Belgique* (2021).

92. Hermans, J., Delaunoy, A., Rozet, F., Wehenkel, A. & Louppe, G. A crisis in simulation-based inference? beware, your posterior approximations can be unfaithful. *Transactions on Machine Learning Research* (2022).

93. Ramaswamy, S., Courcol, J.-D., Abdellah, M., Adaszewski, S. R., Antille, N., Arsever, S., Atenekeng, G., Bilgili, A., Brukau, Y., Chalimourda, A., *et al.* The neocortical microcircuit collaboration portal: a resource for rat somatosensory cortex. *Frontiers in neural circuits* **9**, 44 (2015).

94. Van Geit, W., Gevaert, M., Chindemi, G., Rössert, C., Courcol, J.-D., Muller, E. B., Schürmann, F., Segev, I. & Markram, H. BluePyOpt: leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. *Frontiers in neuroinformatics* **10**, 17 (2016).

95. Pacilio, C., Bhagwat, S. & Cotesta, R. Simulation-based inference of black hole ringdowns in the time domain. *Physical Review D* **110**, 083010 (2024).

96. Miller, T. B., Pasha, I., Polzin, A. & van Dokkum, P. Silkscreen: Direct Measurements of Galaxy Distances from Survey Image Cutouts. *arXiv preprint arXiv:2407.04091* (2024).

97. Araujo, C. P., Ronchi, M., Graber, V. & Rea, N. Radio pulsar population synthesis with consistent flux measurements using simulation-based inference. *arXiv preprint arXiv:2412.04070* (2024).

98. Bernardo, D., Xie, X., Verma, P., Kim, J., Liu, V., Numis, A. L., Wu, Y., Glass, H. C., Yap, P.-T., Nagarajan, S. S., *et al.* Simulation-based inference of developmental EEG maturation with the spectral graph model. *Communications Physics* **7**, 255 (2024).

99. Akhmetzhanova, A., Mishra-Sharma, S. & Dvorkin, C. Data compression and inference in cosmology with self-supervised machine learning. *Monthly Notices of the Royal Astronomical Society* **527**, 7459 (2024).

100. Kaiser, J., Stock, R., Müller, E., Schemmel, J. & Schmitt, S. Simulation-based inference for model parameterization on analog neuromorphic hardware. *Neuromorphic Computing and Engineering* **3**, 044006 (2023).

101. Gao, R., Deistler, M. & Macke, J. H. Generalized Bayesian inference for scientific simulators via amortized cost estimation. *Advances in Neural Information Processing Systems* **36** (2024).

102. Gao, R., Deistler, M. & Macke, J. H. Generalized Bayesian Inference for Scientific Simulators via Amortized Cost Estimation. *arXiv preprint arXiv:2305.15208* (2023).

103. Walker, S. G. Bayesian inference with misspecified models. *Journal of statistical planning and inference* **143**, 1621 (2013).

104. Bissiri, P. G., Holmes, C. C. & Walker, S. G. A general framework for updating belief distributions. *Journal of the royal statistical society. series b, statistical methodology* **78**, 1103 (2016).

105. Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B. & Smola, A. A kernel two-sample test. *The Journal of Machine Learning Research* **13**, 723 (2012).

106. For Brain Science, A. I. Allen cell types database. *http://celltypes.brain-map.org/* (2016).

107. Teeter, C., Iyer, R., Menon, V., Gouwens, N., Feng, D., Berg, J., Szafer, A., Cain, N., Zeng, H., Hawrylycz, M., Koch, C. & Mihalas, S. Generalized leaky integrate-and-fire models classify multiple neuron types. *Nature communications* **9**, 709 (2018).

108. Pospischil, M., Toledo-Rodriguez, M., Monier, C., Piwkowska, Z., Bal, T., Frégnac, Y., Markram, H. & Destexhe, A. Minimal Hodgkin–Huxley type models for different classes of cortical and thalamic neurons. *Biological cybernetics* **99**, 427 (2008).

109. Ward, D., Cannon, P., Beaumont, M., Fasiolo, M. & Schmon, S. Robust neural posterior estimation and statistical model criticism. *Advances in Neural Information Processing Systems* **35**, 33845 (2022).

110. Schmitt, M., Bürkner, P.-C., Köthe, U. & Radev, S. T. *Detecting model misspecification in amortized Bayesian inference with neural networks* in *DAGM German Conference on Pattern Recognition* (2023), 541.

111. Huang, D., Bharti, A., Souza, A., Acerbi, L. & Kaski, S. Learning robust statistics for simulation-based inference under model misspecification. *Advances in Neural Information Processing Systems* **36**, 7289 (2023).

112. Wehenkel, A., Gamella, J. L., Sener, O., Behrmann, J., Sapiro, G., Cuturi, M. & Jacobsen, J.-H. Addressing Misspecification in Simulation-based Inference through Data-driven Calibration. *arXiv preprint arXiv:2405.08719* (2024).

113. Tejero-Cantero, A., Boelts, J., Deistler, M., Lueckmann, J.-M., Durkan, C., Gonçalves, P. J., Greenberg, D. S. & Macke, J. H. SBI–A toolkit for simulation-based inference. *arXiv preprint arXiv:2007.09114* (2020).

114. Hermans, J., Delaunoy, A., Rozet, F., Wehenkel, A., Begy, V. & Louppe, G. A Crisis In Simulation-Based Inference? Beware, Your Posterior Approximations Can Be Unfaithful. *Transactions on Machine Learning Research* (2022).

115. Linhart, J., Gramfort, A. & Rodrigues, P. L-c2st: Local diagnostics for posterior approximations in simulation-based inference. *Advances in Neural Information Processing Systems* **36** (2024).

116. Lemos, P., Coogan, A., Hezaveh, Y. & Perreault-Levasseur, L. *Sampling-based accuracy testing of posterior estimators for general inference* in *International Conference on Machine Learning* (2023), 19256.

117. Bondarenko, V., Nikolaev, M., Kromm, D., Belousov, R., Wolny, A., Blotenburg, M., Zeller, P., Rezakhani, S., Hugger, J., Uhlmann, V., *et al.* Embryo-uterine interaction coordinates mouse embryogenesis during implantation. *The EMBO Journal* **42**, e113280 (2023).

118. Avecilla, G., Chuong, J. N., Li, F., Sherlock, G., Gresham, D. & Ram, Y. Neural networks enable efficient and accurate simulation-based inference of evolutionary parameters from adaptation dynamics. *PLoS biology* **20**, e3001633 (2022).

119. Lowet, E., Sheehan, D. J., Chialva, U., Pena, R. D. O., Mount, R. A., Xiao, S., Zhou, S. L., Tseng, H.-a., Gritton, H., Shroff, S., *et al.* Theta and gamma rhythmic coding through two spike output modes in the hippocampus during spatial navigation. *Cell reports* **42** (2023).

120. Hashemi, M., Vattikonda, A. N., Jha, J., Sip, V., Woodman, M. M., Bartolomei, F. & Jirsa, V. K. Amortized Bayesian inference on generative dynamical network models of epilepsy using deep neural density estimators. *Neural Networks* **163**, 178 (2023).

121. Hahn, C. & Melchior, P. Accelerated Bayesian SED modeling using amortized neural posterior estimation. *The Astrophysical Journal* **938**, 11 (2022).

122. Dingeldein, L., Cossio, P. & Covino, R. Simulation-based inference of single-molecule force spectroscopy. *Machine Learning: Science and Technology* **4**, 025009 (2023).

123. Boelts, J., Harth, P., Gao, R., Udvary, D., Yáñez, F., Baum, D., Hege, H.-C., Oberlaender, M. & Macke, J. H. Simulation-based inference for efficient identification of generative models in computational connectomics. *PLOS Computational Biology* **19**, e1011406 (2023).

124. Glöckler, M., Deistler, M. & Macke, J. H. *Variational methods for simulation-based inference* in *International Conference on Learning Representations* (2022).

125. Muratore, F., Gruner, T., Wiese, F., Belousov, B., Gienger, M. & Peters, J. *Neural posterior domain randomization* in *Conference on Robot Learning* (2022), 1532.

126. Dyer, J., Cannon, P., Farmer, J. D. & Schmon, S. M. *Calibrating Agent-based Models to Microdata with Graph Neural Networks* in *ICML 2022 Workshop AI for Agent-Based Modelling* (2022).

127. Wiqvist, S., Frellsen, J. & Picchini, U. Sequential neural posterior and likelihood approximation. *arXiv preprint arXiv:2102.06522* (2021).

128. Spurio Mancini, A., Docherty, M., Price, M. & McEwen, J. Bayesian model comparison for simulation-based inference. *RAS Techniques and Instruments* **2**, 710 (2023).

129. Dirmeier, S., Albert, C. & Perez-Cruz, F. Simulation-based inference using surjective sequential neural likelihood estimation. *arXiv preprint arXiv:2308.01054* (2023).

130. Boelts, J., Lueckmann, J.-M., Gao, R. & Macke, J. H. Flexible and efficient simulation-based inference for models of decision-making. *Elife* **11**, e77220 (2022).

131. Pyapali, G. K., Sik, A., Penttonen, M., Buzsaki, G. & Turner, D. A. Dendritic properties of hippocampal CA1 pyramidal neurons in the rat: intracellular staining in vivo and in vitro. *Journal of Comparative Neurology* **391**, 335 (1998).

132. Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S. & Zhang, Q. *JAX: composable transformations of Python+NumPy programs* version 0.3.13. 2018.

133. Ascoli, G. A., Donohue, D. E. & Halavi, M. NeuroMorpho. Org: a central resource for neuronal morphologies. *Journal of Neuroscience* **27**, 9247 (2007).

134. Kumbhar, P., Hines, M., Fouriaux, J., Ovcharenko, A., King, J., Delalondre, F. & Schürmann, F. CoreNEURON: an optimized compute engine for the NEURON simulator. *Frontiers in neuroinformatics* **13**, 63 (2019).

135. Ben-Shalom, R., Ladd, A., Artherya, N. S., Cross, C., Kim, K. G., Sanghevi, H., Korngreen, A., Bouchard, K. E. & Bender, K. J. NeuroGPU: Accelerating multi-compartment, biophysically detailed neuron simulations on GPUs. *Journal of neuroscience methods* **366**, 109400 (2022).

136. Zhang, Y., He, G., Ma, L., Liu, X., Hjorth, J. J., Kozlov, A., He, Y., Zhang, S., Kotaleski, J. H., Tian, Y., *et al.* A GPU-based computational framework that bridges Neuron simulation and Artificial Intelligence. *Nature Communications* **14**, 5798 (2023).

137. Griewank, A. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and software* **1**, 35 (1992).

138. Jones, I. S. & Kording, K. P. Efficient optimization of ODE neuron models using gradient descent. *arXiv preprint arXiv:2407.04025* (2024).

139. Van Geit, W., De Schutter, E. & Achard, P. Automated neuron model optimization techniques: a review. *Biological cybernetics* **99**, 241 (2008).

140. Rajan, K., Harvey, C. D. & Tank, D. W. Recurrent network models of sequence generation and memory. *Neuron* **90**, 128 (2016).

141. Schröder, C., Klindt, D., Strauss, S., Franke, K., Bethge, M., Euler, T. & Berens, P. System identification with biophysical constraints: A circuit model of the inner retina. *Advances in Neural Information Processing Systems* **33**, 15439 (2020).

142. Xu, Y., Zou, P. & Cohen, A. E. Voltage imaging with genetically encoded indicators. *Current Opinion in Chemical Biology* **39**, 1 (2017).

143. Magee, J. C. Dendritic hyperpolarization-activated currents modify the integrative properties of hippocampal CA1 pyramidal neurons. *Journal of Neuroscience* **18**, 7613 (1998).

144. Huys, Q. J., Ahrens, M. B. & Paninski, L. Efficient estimation of detailed single-neuron models. *Journal of neurophysiology* **96**, 872 (2006).

145. Schwartz, G. W., Okawa, H., Dunn, F. A., Morgan, J. L., Kerschensteiner, D., Wong, R. O. & Rieke, F. The spatial structure of a nonlinear receptive field. *Nature neuroscience* **15**, 1572 (2012).

146. Fohlmeister, J. & Miller, R. Impulse encoding mechanisms of ganglion cells in the tiger salamander retina. *Journal of neurophysiology* **78**, 1935 (1997).

147. Ran, Y., Huang, Z., Baden, T., Schubert, T., Baayen, H., Berens, P., Franke, K. & Euler, T. Type-specific dendritic integration in mouse retinal ganglion cells. *Nature Communications* **11**, 2101 (2020).

148. McIntosh, L., Maheswaranathan, N., Nayebi, A., Ganguli, S. & Baccus, S. Deep learning models of the retinal response to natural scenes. *Advances in neural information processing systems* **29** (2016).

149. Idrees, S., Manookin, M. B., Rieke, F., Field, G. D. & Zylberberg, J. Biophysical neural adaptation mechanisms enable artificial neural networks to capture dynamic retinal computation. *Nature Communications* **15**, 5957 (2024).

150. Qiu, Y., Klindt, D. A., Szatko, K. P., Gonschorek, D., Hoefling, L., Schubert, T., Busse, L., Bethge, M. & Euler, T. Efficient coding of natural scenes improves neural system identification. *PLoS computational biology* **19**, e1011037 (2023).

151. Hines, M., Davison, A. P. & Muller, E. NEURON and Python. *Frontiers in neuroinformatics* **3**, 391 (2009).

152. Goodman, D. F. & Brette, R. The brian simulator. *Frontiers in neuroscience* **3**, 643 (2009).

153. Yu, Y., Xia, R., Ma, Q., Lengyel, M. & Hennequin, G. *Second-order forward-mode optimization of recurrent neural networks for neuroscience* in *The Thirty-eighth Annual Conference on Neural Information Processing Systems* (2024).

154. Wong-Campos, J. D., Park, P., Davis, H., Qi, Y., Tian, H., Itkis, D. G., Kim, D., Grimm, J. B., Plutkis, S. E., Lavis, L., *et al.* Voltage dynamics of dendritic integration and back-propagation in vivo. *bioRxiv* (2023).

155. Betancourt, M. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434* (2017).

156. Lappalainen, J. K., Tschopp, F. D., Prakhya, S., McGill, M., Nern, A., Shinomiya, K., Takemura, S.-y., Gruntman, E., Macke, J. H. & Turaga, S. C. Connectome-constrained deep mechanistic networks predict neural responses across the fly visual system at single-neuron resolution. *bioRxiv*, 2023 (2023).

157. Daxberger, E., Kristiadi, A., Immer, A., Eschenhagen, R., Bauer, M. & Hennig, P. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems* **34**, 20089 (2021).

158. Lückmann, J.-M. *Simulation-based inference for neuroscience and beyond* PhD thesis (Universität Tübingen, 2022).

159. Watson, K. K., Jones, T. K. & Allman, J. M. Dendritic architecture of the von Economo neurons. *Neuroscience* **141**, 1107 (2006).

APPENDIX

A PUBLICATIONS

This thesis includes the following articles which have been published in peer-reviewed journals or conferences:

1. M. Deistler, J. H. Macke[‡] & P. J. Gonçalves[‡], Energy-efficient network activity from disparate circuit parameters. *Proceedings of the National Academy of Sciences* **119**, e2207632119 (2022).

2. M. Deistler, P. J. Goncalves[‡], & J. H. Macke[‡], Truncated proposals for scalable and hassle-free simulation-based inference. *Advances in Neural Information Processing Systems* **35**, 23135 (2022).

3. R. Gao[†], M. Deistler[†] & J. H. Macke, Generalized Bayesian inference for scientific simulators via amortized cost estimation. *Advances in Neural Information Processing Systems* **36** (2024).

4. Á. Tejero-Cantero[†], J. Boelts[†], M. Deistler[†], J.-M. Lueckmann[†], C. Durkan[†], P. J. Gonçalves, D. S. Greenberg & J. H. Macke, sbi: A toolkit for simulation-based inference. *Journal of Open Source Software* **5**, 2505 (2020).

In addition, it includes the following pre-print articles:

5. J. Boelts[†], M. Deistler[†], M. Gloeckler, Á. Tejero-Cantero, J.-M. Lueckmann, G. Moss, P. Steinbach, T. Moreau, F. Muratore, J. Linhart et al., sbi reloaded: a toolkit for simulation-based inference workflows. *arXiv preprint arXiv:2411.17337* (2024).

6. M. Deistler, K. L. Kadhim, M. Pals, J. Beck, Z. Huang, M. Gloeckler, J. K. Lappalainen, C. Schröder, P. Berens, P. J. Gonçalves & J. H. Macke, Differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics. *bioRxiv*, 2024 (2024).

[†] indicates shared first authorship.
[‡] indicates shared last authorship.

Below, we provide copies of these publications and pre-print articles.

PNAS RESEARCH ARTICLE | NEUROSCIENCE

Energy-efficient network activity from disparate circuit parameters

Michael Deistler[a], Jakob H. Macke[a,b,1,2], and Pedro J. Gonçalves[a,c,1,2]

Edited by Terrence Sejnowski, Salk Institute for Biological Studies, La Jolla, CA; received May 5, 2022; accepted September 19, 2022

Neural circuits can produce similar activity patterns from vastly different combinations of channel and synaptic conductances. These conductances are tuned for specific activity patterns but might also reflect additional constraints, such as metabolic cost or robustness to perturbations. How do such constraints influence the range of permissible conductances? Here we investigate how metabolic cost affects the parameters of neural circuits with similar activity in a model of the pyloric network of the crab *Cancer borealis*. We present a machine learning method that can identify a range of network models that generate activity patterns matching experimental data and find that neural circuits can consume largely different amounts of energy despite similar circuit activity. Furthermore, a reduced but still significant range of circuit parameters gives rise to energy-efficient circuits. We then examine the space of parameters of energy-efficient circuits and identify potential tuning strategies for low metabolic cost. Finally, we investigate the interaction between metabolic cost and temperature robustness. We show that metabolic cost can vary across temperatures but that robustness to temperature changes does not necessarily incur an increased metabolic cost. Our analyses show that despite metabolic efficiency and temperature robustness constraining circuit parameters, neural systems can generate functional, efficient, and robust network activity with widely disparate sets of conductances.

energy efficiency | neuronal variability | neural dynamics | simulation-based inference | Bayesian inference

Significance

Neural systems have the remarkable feature of showing similar activity patterns despite having disparate underlying mechanistic properties (e.g., different ion channel densities). This feature, called parameter degeneracy, underlies the capacity of neural systems to compensate for perturbations to their components. Less understood is whether parameter degeneracy is reduced (or eliminated) by biological constraints, notably to preserve metabolic efficiency or robustness to environmental fluctuations. Developing machine learning methods for degeneracy analysis, we investigated this question in a computational model of the pyloric circuit in the crab stomatogastric ganglion. We found that this circuit can generate functional, energy-efficient, and temperature-robust activity with widely disparate sets of membrane and synaptic conductances. This approach could be exploited beyond the current system.

Neural activity arises from the interplay of mechanisms at multiple levels, including single-neuron and network mechanisms. Several experimental and theoretical studies have found that neural systems can produce similar activity from vastly different membrane and synaptic conductances (1–6), a property sometimes referred to as parameter degeneracy (7, 8). Such parameter degeneracy has been argued to be a prerequisite for natural selection (7) and translates into potential mechanisms of compensation for perturbations of the systems' parameters (3, 5, 9–14). However, in addition to a specific target activity, neural systems are likely subject to additional constraints such as the requirement to be energy efficient (15–17). In order to understand experimentally observed variability and probe potential compensation mechanisms in functioning neural systems, it is thus crucial to characterize the extent of the systems' parameter degeneracy under such additional constraints.

Neuronal activity accounts for the majority of the energy consumed by the brain (18–20). Energy is stored in the ionic gradients across the cell membrane and consumed mostly by action potentials and synaptic mechanisms. Maintaining the ionic gradients requires the action of ion pumps, which consume ATP (15, 21). Previous work has investigated the metabolic efficiency in small neural systems, often at the single-neuron level and with few ion channels (often sodium, potassium, and leak) (15, 22, 23). In these studies, it has been demonstrated that energy consumption of single neurons can be reduced by tuning maximal conductances or time constants of gating variables, while maintaining electrophysiological characteristics, e.g., spike width. However, questions regarding energy efficiency of neural systems remain: First, it is unclear whether previous findings in single neurons (24–26) extrapolate to neural circuits with a large diversity of membrane and synaptic currents (12, 21, 27). Second, the question of how strongly metabolic constraints impact parameter degeneracy remains unaddressed: are energy efficient solutions confined in parameter space, or can disparate network parameters generate energy efficient activity? Last, metabolic cost is only one of many constraints under which neural circuits operate, and it is often unknown whether energy efficiency trades off with other constraints [for a study of how energy efficiency trades off with temperature robustness in a single neuron model of the grasshopper, see Roemschied et al. (28)].

Here we investigate how energy efficiency constrains the parameter degeneracy in the pyloric network in the stomatogastric ganglion of the crab *Cancer borealis* (29, 30), a canonical example of a neural system with parameter degeneracy (5). The pyloric

Author contributions: M.D., J.H.M., and P.J.G. designed research; M.D., J.H.M., and P.J.G. performed research; M.D. contributed new reagents/analytic tools; M.D. analyzed data; and M.D., J.H.M., and P.J.G. wrote the paper.

The authors declare no competing interest.

This article is a PNAS Direct Submission.

[1]J.H.M. and P.J.G. contributed equally to this work.

[2]To whom correspondence may be addressed. Email: jakob.macke@uni-tuebingen.de or pedro.goncalves@uni-tuebingen.de.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2207632119/-/DCSupplemental.

Published October 24, 2022.

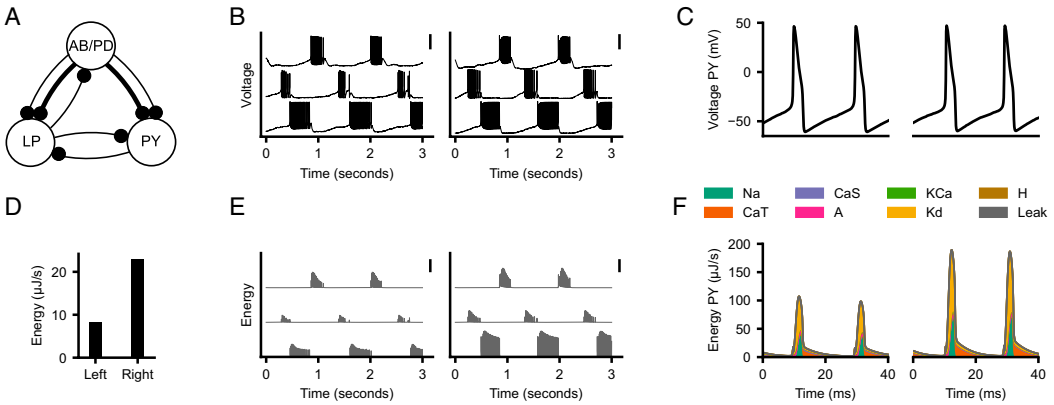


Fig. 1. Similar activity with different energy consumption. (*A*) Computational model of the pyloric network consisting of three model neurons (AB/PD, LP, and PY) and seven synapses. (*B*) Two model configurations with similar circuit activity (traces from top to bottom, AB/PD, LP, and PY) despite different circuit parameters (parameter values not plotted). (Scale bars, 50 mV.) (*C*) Close-up of two spikes in the PY neural activity shown in *B*. (*D*) Total energy consumption divided by the duration of the simulation (10 s) for the traces shown in *B*. The left circuit has threefold lower metabolic cost than the right circuit. (*E*) Consumed energy at each time point. (Scale bar, 100 μJ/s.) (*F*) Energy consumed by each of the ion currents during the two spikes shown in *C* (stacked vertically).

network produces a triphasic motor pattern and consists of a pacemaker kernel (anterior burster neuron [AB] and two pyloric dilator neurons [PD]), as well as two types of follower neurons (a single lateral pyloric [LP] and several pyloric [PY] neurons), interconnected by inhibitory synapses. A model of this circuit with three model neurons (AB/PD, LP, and PY), each with eight membrane currents, and seven inhibitory synapses (Fig. 1*A*; details in *SI Appendix*, SI Text) has been shown to be capable of producing similar network activity with widely different parameters (5).

We start by characterizing the parameter degeneracy of this model: We apply a recently introduced machine learning tool for simulation-based inference, sequential neural posterior estimation (SNPE) (14), to estimate the full set of membrane and synaptic conductances for which the model reproduces experimentally measured electrophysiological activity. We reduce the number of model simulations required to run SNPE by introducing an additional classifier which detects and rejects parameter combinations that produce nonbursting model outputs (31). After characterizing the parameter degeneracy in the model, we show that disparate circuit configurations can have different energy consumption despite similar activity. However, a significant parameter degeneracy is present in the model even when enforcing circuits to have both similar activity and low energy consumption. Furthermore, energy consumption is linearly predictable from circuit parameters, allowing us to identify tuning mechanisms for low metabolic cost. We then show that individual neurons in the pyloric network can be tuned separately to minimize their energy consumption and thereby achieve low energy consumption at the circuit level. Finally, since the crab *C. borealis* is subject to daily and seasonal fluctuations in temperature, we study the trade-off between metabolic cost and robustness to changes in temperature (32–35). We find that metabolic cost can vary across temperatures but that the pyloric network can produce functional, energy-efficient, and temperature-robust activity with disparate parameters.

Results

Disparate Energy Consumption despite Similar Network Activity. We studied the metabolic cost in a model of the pyloric network (Fig. 1*A*). In this model, disparate sets of maximal membrane and synaptic conductances can give rise to similar network activity (5). As an example, we simulated two such circuit configurations (Fig. 1*B*) and computed their metabolic cost using a previously described measure of energy consumption (36). In this measure, the energy for each ion channel is the time integral of the product of the membrane current and the respective difference between the membrane voltage and the reversal potential (details in *SI Appendix*, SI Text). The energy consumed by the entire neural circuit is the sum of the energies across channels of all neurons. We note that this power-based energy measure likely underestimates the true energy consumption but is strongly correlated with the current-based energy measure based on sodium and calcium currents (15, 25) (*SI Appendix*, Figs. S1 and S2). Since the power-based measure naturally allows us to quantify the energy consumed by a larger diversity of currents, we performed our main analyses with this measure, although our main results are maintained when using the current-based measure (*SI Appendix*, Fig. S3).

Although the two simulated circuit configurations produce similar network activity, even at the single-spike level (Fig. 1*C*), the total energy consumption (Fig. 1*D*) as well as the moment by moment energy consumption differ substantially (Fig. 1*E*). A closer inspection of the energy consumed by each current in the PY neuron during the action potentials (37) shows that the difference in energy between these two network configurations is also evident in the energy consumed by the sodium current Na, the delayed-rectifier potassium current K_d, and the transient calcium current CaT (Fig. 1*F*).

Disparity in Energy Consumption in Models Matching Experimental Data. The example above illustrates that the model of the pyloric network can, in principle, produce the same activity with different metabolic costs. However, it is unclear how broad the range of metabolic costs associated with the same network output is. In order to address this, we need to identify the full space of maximal membrane and synaptic conductances (31 parameters in total) that match experimental measurements of network activity and to characterize the energy consumption of each of these configurations.

We used a recently introduced machine learning tool for simulation-based inference, SNPE (14), to estimate the set of circuit parameters (the posterior distribution) consistent with data

and prior assumptions about the parameters. In SNPE, parameters which specify network configurations are initially sampled from the prior distribution (in our case, a uniform distribution within plausible parameter ranges) and used to simulate network activity. Subsequently, a neural network–based density estimator is trained on these simulated network activities to learn which parameter sets produce network activity that is compatible with empirical observations.

In order to generate the training data for the neural network, SNPE requires millions of model simulations to accurately infer the set of data-compatible parameters. To improve the simulation efficiency and make the neural network predict parameter sets that more closely match experimental data, we introduced a modification of the algorithm (Fig. 2*A*). Specifically, a technical challenge for SNPE is that parameter sets sampled from the prior distribution might produce simulation results that are not valid, i.e., produce clearly nonsensible data: for example, if there are no bursts, phase gaps between bursts are not defined (Fig. 2*A*, fourth panel, red). For SNPE, these invalid simulations are discarded immediately. In order to reduce the fraction of simulations that are discarded, we introduce a classifier to predict whether a parameter set will lead to a valid or an invalid simulation output (31) (Fig. 2*A*, second panel). Once the classifier is trained on an initial set of simulations, parameters are immediately discarded without running the simulation, if the classifier confidently predicts that the simulation will be invalid (details in *SI Appendix*, SI Text). We name the distribution of parameters that are accepted by the classifier the "restricted prior" (Fig. 2*A*, third panel). Once sufficiently many valid simulations are performed, SNPE proceeds by training a deep neural density estimator to estimate the posterior distribution over parameters of the model (14) (Fig. 2*A*, fifth and sixth panels; proof of convergence to the correct posterior distribution in *SI Appendix*, SI Text).

We used this procedure to infer the posterior distribution over maximal membrane and synaptic conductances of the model of the pyloric network given salient and physiologically relevant features of experimentally measured data. These features are the cycle period, burst durations, duty cycles, phase gaps, and phase delays of the triphasic rhythm (Fig. 2*B*; details in *SI Appendix*, SI Text) (38). As in previous studies (4, 5), we did

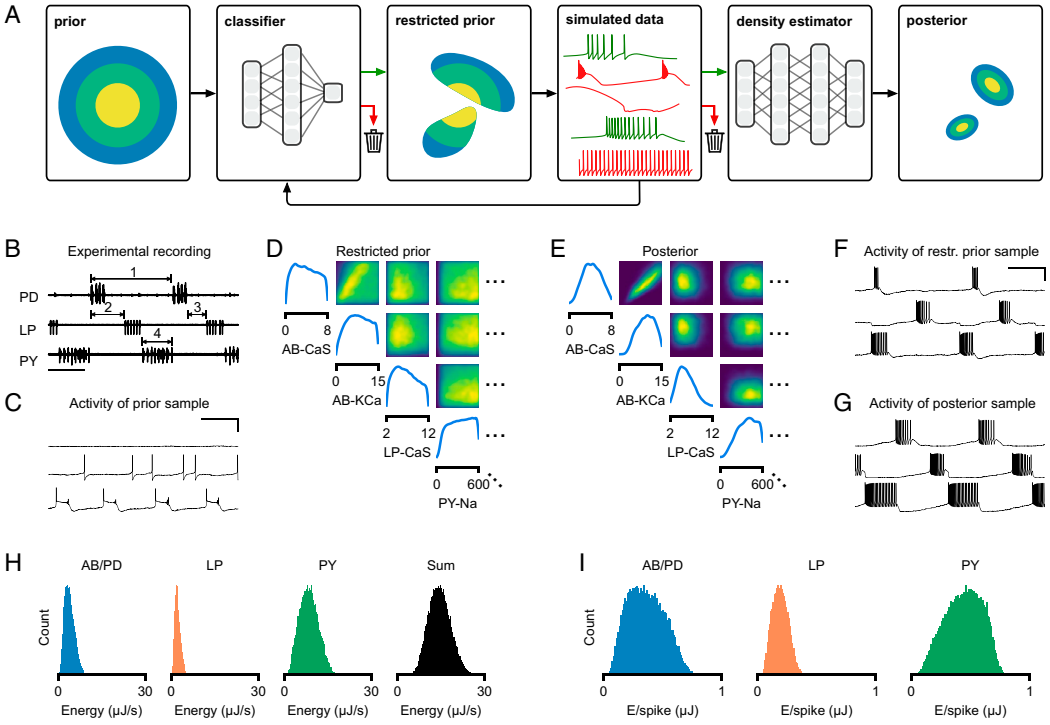


Fig. 2. Bayesian inference reveals wide range of energy consumption. (*A*) Inferring the posterior distribution by combining a rejection classifier and a deep neural density estimator. First, a classifier (trained on an initial set of simulations) predicts which circuit parameters sampled from the prior produce valid simulation outputs. We then proceed by sampling from the part of the parameter space that is accepted by the classifier, i.e., the restricted prior. All valid data (green) are used to train a deep neural density estimator, and all invalid data are discarded (red) (14). Once this estimator is trained, it can be evaluated on experimental data to return the posterior distribution over model parameters. (*B*) Experimental data recorded from the pyloric network (38). Arrows indicate a subset of the physiologically relevant features, namely, the cycle period (arrow 1), phase delays (arrow 2), phase gaps (arrow 3), and burst durations (arrow 4) (see *SI Appendix*, SI Text for details). (Scale bar, 500 ms.) (*C*) Simulation output from a parameter set sampled from the prior distribution. The traces are AB/PD (*Top*), LP (*Middle*), and PY (*Bottom*). (Scale bars, 500 ms and 50 mV.) (*D*) Subset of the marginals and pairwise marginals of the 31-dimensional restricted prior, i.e., the subspace of parameters for which the model produces bursting activity. All maximal conductances are given in mS/cm^2. (*E*) Subset of the marginals and pairwise marginals of the posterior distribution, i.e., the subspace of parameters for which the model matches experimental data shown in *B* (full posterior distribution in *SI Appendix*, Fig. S13). (*F*) Sample from the restricted prior producing bursting activity but not matching experimental data. (*G*) Sample from the posterior distribution closely matching features of the experimental data. (*H*) Histograms over energy consumed by each neuron (blue, orange, and green) as well as by the entire circuit (black), divided by the duration of the simulation. Trace with lowest energy consumes nine times less energy than trace with highest energy. (*I*) Same as in *H* but for energy per spike.

not constrain the model of the pyloric network by the number of spikes or the spike shapes. Below, we describe the results obtained for a specific experimental preparation. We qualitatively reproduced all results with two additional experimental preparations (*SI Appendix*, Figs. S4–S9) (38). We note that SNPE captures parameter uncertainty stemming from epistemic uncertainty as well as from nonidentifiability of parameters (i.e., parameter degeneracy). Given that the experimentally measured voltage traces contain little variability between different cycles of the triphasic rhythm (*SI Appendix*, Fig. S10), we introduced a modest amount of current noise in the pyloric network model (*SI Appendix*, Fig. S11; details in *SI Appendix*, SI Text). Thus, the epistemic uncertainty of the inference is small, and posterior uncertainty is likely dominated by parameter degeneracy. Further evidence of this is the fact that different noise sources do not significantly affect posterior variability (*SI Appendix*, Fig. S12).

When simulating the pyloric network model with parameter sets sampled from the prior distribution, 99% of simulations do not produce spikes or bursts, and hence, characteristic summary features of the circuits are not defined (Fig. 2*C*). The restricted prior (Fig. 2*D*) is narrower than the prior distribution but considerably broader than the posterior (Fig. 2*E*; full posterior distribution in *SI Appendix*, Fig. S13; comparison between prior, restricted prior, and posterior in *SI Appendix*, Fig. S14). Parameters sampled from the restricted prior often produce activity with well-defined summary features (Fig. 2*F*) but do not generally match experimental data, whereas samples from the posterior closely match experimental data (Fig. 2*G*). By using the classifier to reject invalid simulations, we required half as many simulations compared to classical SNPE (14) and achieved a higher accuracy (*SI Appendix*, Fig. S15). For the subsequent analyses, we only considered posterior samples whose activity was within a prescribed distance to the experimental data and discarded all other samples (details in *SI Appendix*, SI Text). We simulated 1 million parameter configurations sampled from the posterior, out of which ∼3.5% fulfilled the distance criterion, leading to a database of 35,939 parameter sets whose activity closely matched experimental data. Sampling from the prior distribution rather than the posterior would have required ∼600 billion simulations to obtain 35,939 parameter sets that fulfill our criterion (60,000 times more than with our method).

We computed the energy consumption of each of the 35,939 circuit activities (Fig. 2*H*). The circuit configuration with lowest total energy consumes nine times less energy than the circuit configuration with highest total energy. To ensure that the difference in energy does not only stem from different numbers of spikes within a burst, we also computed the average energy consumed during a spike (energy per spike) in each of the neurons (Fig. 2*I*). As with total energy, energy per spike strongly varies across parameter configurations. We note that the differences between the individual neurons might in part be attributed to the different prior ranges of maximal conductances across neurons [ranges based on results from Prinz et al. (5); *SI Appendix*, SI Text]. These results show that despite similar circuit function, different parameter sets can have vastly different energy consumption. Below, we investigate the mechanisms giving rise to this phenomenon.

Metabolic Constraints on Individual Circuit Parameter Ranges. How strongly does enforcing low energy consumption constrain the permissible ranges of circuit parameters? We inspected the circuit parameters of the 2% most and least efficient configurations within our database of 35,939 model configurations (Fig. 3 *A*, *Left*). For some circuit parameters, the range of values producing efficient activity is clearly different from the range of

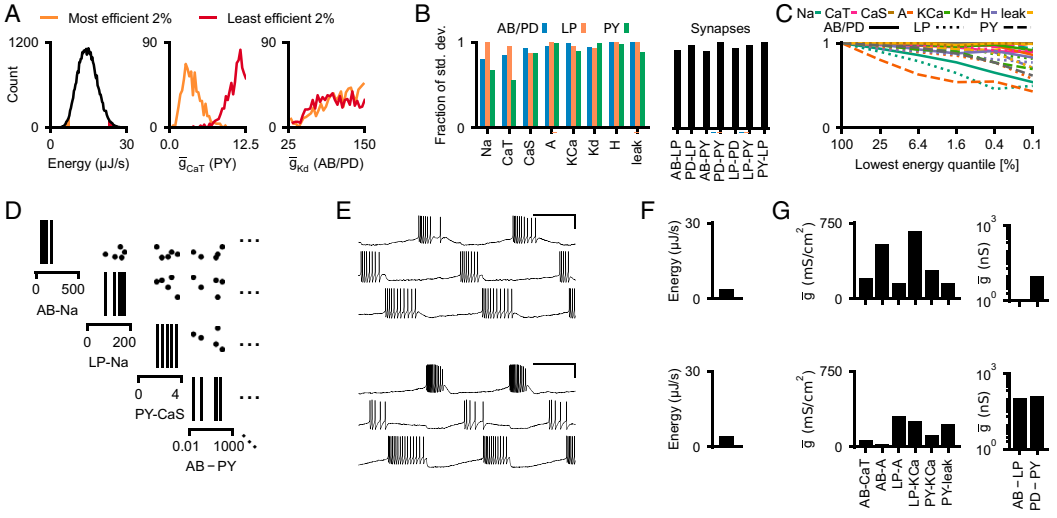


Fig. 3. Metabolic constraints on individual circuit parameters. (*A*) (*Left*) Time-averaged energy consumption of 35,939 models that match experimental data. The orange area corresponds to the energy consumption in the lowest 2% quantile, and the red area corresponds to the top 98% quantile. (*Middle*) Distribution of the maximal conductance of the transient calcium channel (CaT) in the PY neuron in the 2% (orange) and 98% quantile (red). (*Right*) Distribution of the maximal conductance of the delayed-rectifier potassium channel (K_d) in the AB/PD neuron in the 2% (orange) and 98% quantile (red). (*B*) SD of parameters for models with energy consumption in the lowest 2% quantile. SD is normalized to the SD of the parameters across all 35,939 models in our database. (*C*) Same as *B* but for a range of quantiles. Solid, AB/PD; dotted, LP; dashed, PY. Colors are the same as in Fig. 1*F*. Synapses are in *SI Appendix*, Fig. S18. (*D*) Subset of the parameter values of the five most efficient circuit configurations in our database. (*E*) The network activity produced by two of these five configurations. (Scale bar, 500 ms and 50 mV.) (*F*) Time-averaged energy consumption of the two configurations shown in *D*. (*G*) Subset of circuit parameters of the two solutions shown in *B*. Despite similar network activity and low energy consumption, several parameters differ by more than twofold. The membrane conductances are scaled by the following factors (left to right): 100, 10, 10, 100, 100, and 10,000.

values producing energetically costly activity (e.g., the maximal conductance of the transient calcium current in the PY neuron; Fig. 3 *A*, *Middle*). For other parameters, the range does not change (e.g., the maximal conductance of the delayed-rectifier potassium current in the AB/PD neuron; Fig. 3 *A*, *Right*). To quantify how strongly low energy consumption constrains parameters, we compared the parameter SD across all 35,939 model configurations to that of the most efficient 2% (Fig. 3 *B* and *C*; *SI Appendix*, Fig. S16 show how the parameter means shift as energy constraints are enforced). Most parameters in the circuit barely get constrained by energy consumption (values close to 1 in Fig. 3 *B* and *C*). The parameters that get constrained the most by enforcing low energy consumption are the Na and CaT conductances of the AB/PD neuron; the CaS conductance of the LP neuron; and the Na, CaT, CaS, and leak conductances of the PY neuron. However, for all of these parameters, a large fraction of variability remains. In addition, we found that enforcing energy constraints affects correlations between parameters only weakly (*SI Appendix*, Fig. S17).

In order to ensure that the remaining variability of circuit parameters does not stem from the remaining variability of energy consumptions within the lowest 2% quantile, we inspected the five most efficient configurations in our database of 35,939 model configurations. Even these five circuit configurations have strongly disparate circuit parameters (Fig. 3*D*). Despite having similar activity (Fig. 3*E*) and very low (and similar) metabolic cost (Fig. 3*F*), their circuit parameters are disparate (Fig. 3*G*). These results demonstrate that metabolic efficiency constrains the range of some circuit parameters, but it is possible to achieve low metabolic cost and similar network activity with widely disparate circuit parameters.

Sodium and Calcium Conductances Influence Energy Consumption Most Strongly. We wanted to understand how each circuit parameter affects energy consumption within our database of 35,939 model configurations. Given that the voltage wave forms of the model configurations within our database are constrained to be similar, energy consumption is expected to be approximately linearly related to the maximal conductances (*SI Appendix*, SI Text). We, therefore, performed a linear regression $E = w^T\theta + b$ from circuit parameters (taken from our database of 35,939 model configurations) onto the energy consumption of these circuits (Fig. 4*A*; details in *SI Appendix*, SI Text). This linear regression achieved a high accuracy, demonstrating that energy consumption can indeed be linearly predicted from circuit parameters (Fig. 4*B*; a nonlinear regression with a neural network leads to similar results and is shown in *SI Appendix*, Fig. S19; details in *SI Appendix*, SI Text). The regression weights w indicate how strongly energy consumption is correlated with each parameter value (Fig. 4*C*). The maximal sodium conductances $\overline{g}_{\mathrm{Na}}$, transient calcium conductances $\overline{g}_{\mathrm{CaT}}$, and slow calcium conductances $\overline{g}_{\mathrm{CaS}}$ are most strongly correlated with energy consumption: increases of these conductances are associated with an increase in energy consumption, and thus, small conductance values correspond to metabolically more efficient solutions. The membrane conductances of the PY neuron influence energy consumption more strongly than the LP and AB/PD neurons. This is expected since the PY model neuron consumes more energy than the other neurons (Fig. 2*H*). The synaptic conductances are weakly correlated with energy consumption, which can be explained by the low values of the maximal synaptic conductances: the synaptic strengths range up to 1,000 nS, whereas the membrane conductances can range

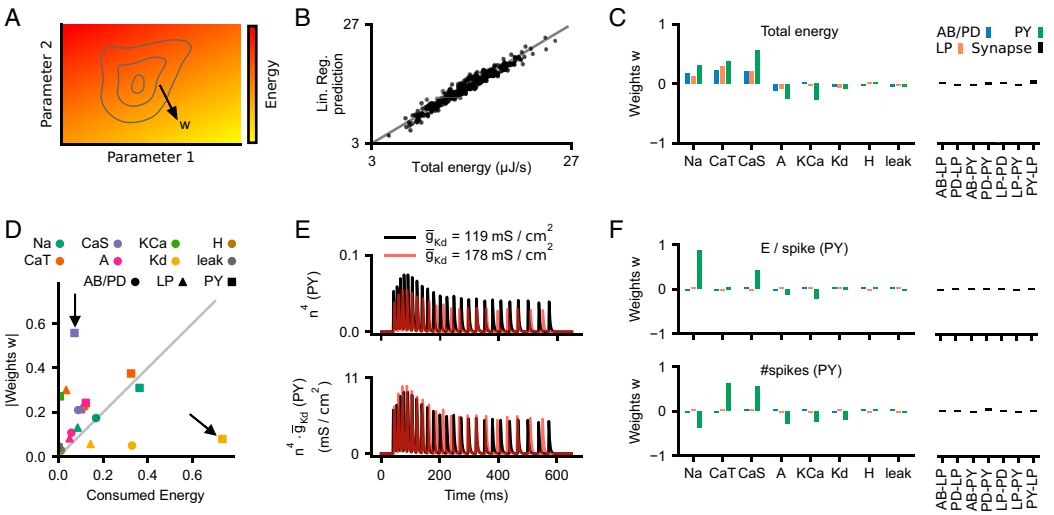


Fig. 4. Influence of circuit parameters on energy consumption. (*A*) Illustration of the energy landscape under functional constraints (matching experimental activity). The linear regression weights correspond to the direction along which energy varies. (*B*) The linear regression accurately predicts the energy consumption on a test set of 300 circuit configurations (black dots). Gray line is the identity function. (*C*) Weights w of the linear regression. (*Left*) Weights of the maximal membrane conductances. (*Right*) Weights of the maximal synaptic conductances. (*D*) Weights w as a function of energy consumption (both normalized), for all membrane currents (arrows highlight two illustrative examples). Membrane conductances on the top left consume little energy, but their maximal conductances correlate strongly with energy consumption. Conductances on the bottom right consume a lot of energy, but their maximal conductances correlate weakly with energy consumption. (*E*) (*Top*) The gating variable n^4 of the K_d current in the PY neuron during activity produced by two circuit configurations (black and red), which are identical apart from the magnitude of $\overline{g}_{\mathrm{Kd}}$. (*Bottom*) The product of gating variable and maximal conductance $n^4 \cdot \overline{g}_{\mathrm{Kd}}$ for the same configurations. (*F*) (*Top*) Weights of a linear regression onto the energy per spike in the PY neuron. (*Bottom*) Weights of a linear regression onto the number of spikes in the PY neuron.

up to 0.4 mS (i.e., 4×10^5 nS), such that synapses consume only 0.08% of the total energy in the circuit. These results demonstrate that energy consumption can be linearly predicted from circuit parameters and that energy consumption is most strongly correlated with the maximal conductances of sodium as well as slow and transient calcium.

How do different currents affect total energy consumption? Do they directly consume energy, or do they trigger processes that then require energy? We addressed these questions by comparing the fraction of energy consumed by each current [as defined by our measure of energy (36); Fig. 1*F*] to the linear regression weight w associated with its maximal conductance (Fig. 4*D*). We found that some currents consume a lot of energy, although their maximal conductances barely correlate with energy consumption, e.g., the K_d current in the PY neuron (Fig. 4*D*, bottom right arrow), while other currents consume little energy, but nonetheless their maximal conductances are correlated with energy consumption, e.g., the CaS current of the PY neuron (Fig. 4*D*, top left arrow).

We investigated the neuronal mechanisms that give rise to these behaviors. First, to understand how currents can consume large amounts of energy despite their maximal conductance only weakly correlating with energy, we investigated the effects of the delayed-rectifier potassium conductance $\overline{g}_{\mathrm{Kd}}$ on circuit activity. We simulated two circuit configurations, identical apart from the magnitude of $\overline{g}_{\mathrm{Kd}}$ in the PY model neuron. In the configuration with higher $\overline{g}_{\mathrm{Kd}}$, the gating variable n did not reach values as high as those for the other configuration, thus leading to a similar effective conductance $n^4 \cdot \overline{g}_{\mathrm{Kd}}$ (Fig. 4*E*). This demonstrates that changes in the maximal conductance $\overline{g}_{\mathrm{Kd}}$ only weakly influence the current and thereby the energy consumption. Thus, despite the potassium current consuming a lot of energy due to a large flow of ions (compared to other channels), its maximal conductance $\overline{g}_{\mathrm{Kd}}$ only weakly correlates with energy consumption.

Second, to understand how maximal conductances can correlate with energy consumption despite their channels consuming little energy, we disentangled the correlation of circuit parameters with energy consumption into two parts: the energy per spike and the number of spikes. We fitted two additional linear regression models: one regression from circuit parameters onto number of spikes in the PY neuron and one regression from circuit parameters onto energy per spike in the PY neuron. We again found good predictive performance of these models, showing that the energy per spike and the number of spikes can also be linearly predicted from circuit parameters (regression performance in *SI Appendix*, Fig. S20). The energy per spike is strongly correlated with the sodium conductance (Fig. 4 *F*, *Top*), whereas the number of spikes is most strongly correlated with the maximal conductance of transient and slow calcium (Fig. 4 *F*, *Bottom*). This demonstrates that increases in the maximal conductance of calcium lead to a higher number of spikes, which involve increased energy consumption through other currents. We verified this hypothesis by simulating two configurations that were identical apart from the magnitude of $\overline{g}_{\mathrm{CaS}}$ in the PY model neuron and found that the configuration with higher $\overline{g}_{\mathrm{CaS}}$ indeed produced more spikes per burst (*SI Appendix*, Fig. S21). This shows that despite the slow calcium channel consuming little energy itself, increasing $\overline{g}_{\mathrm{CaS}}$ can lead to higher energy consumption by increasing the number of spikes, which involve energy consumption through other currents (mostly sodium and potassium). Overall, our analyses demonstrate that currents which consume a lot of energy are not necessarily the ones that influence energy the most.

Minimal Tuning Mechanisms for Low Energy Consumption. We identified circuit parameters that correlate with energy consumption, but this does not yet address the question of which changes of these parameters will lead to the reduction of energy consumption: First, a correlation between parameter values and energy consumption does not imply a causal connection between these. Second, parameters that correlate strongly with energy consumption might have to be finely tuned to match the pyloric rhythm, thus not constituting a feasible substrate for reducing energy consumption. Therefore, we went beyond the previous analysis to investigate potential tuning mechanisms involving single and pairs of parameters that would reduce energy consumption while maintaining the pyloric rhythm.

We investigated how strongly energy consumption could be reduced by mechanisms that involve a single parameter. For instance, we kept all parameters but the maximal sodium conductance of the AB/PD neuron ($\overline{g}_{\mathrm{Na}}$) constant and varied $\overline{g}_{\mathrm{Na}}$ on a grid. We then estimated the energy consumption of each configuration with the previously identified linear model (Fig. 4). The energy consumption of the circuit increases with $\overline{g}_{\mathrm{Na}}$ (Fig. 5*A*), but for too low (or too high) $\overline{g}_{\mathrm{Na}}$, the network activity does not match experimental data (we rejected parameters for which the posterior density is too low; *SI Appendix*, SI Text). Thus, despite $\overline{g}_{\mathrm{Na}}$ strongly correlating with energy consumption (Fig. 4*C*), energy consumption can be reduced only modestly when tuning $\overline{g}_{\mathrm{Na}}$ and keeping all other parameters constant.

We then investigated whether pairwise mechanisms could lead to larger savings in energy consumption. For instance, we kept all parameters but $\overline{g}_{\mathrm{Na}}$ and the delayed-rectifier potassium conductance of the AB/PD neuron ($\overline{g}_{\mathrm{Kd}}$) constant and varied the remaining two parameters on a grid. We estimated the energy consumption of any configuration on this grid and found that the most efficient parameter configuration is 23% more efficient than the most wasteful configuration (Fig. 5*B*). This reduction in energy consumption could be achieved through a simple pairwise mechanism: a reduction of sodium combined with an increase of potassium allows the network to maintain its activity (Fig. 5 *C* and *D*), while reducing the metabolic cost (Fig. 5*B*).

We repeated this analysis for every conductance and every pair of conductances (Fig. 5 *E* and *F*). Note that we only considered pairs of parameters within each neuron because pairwise compensation mechanisms across neurons have been shown to be weak in this model (14). Some of the single-conductance mechanisms can reduce the energy consumption by up to 36%. Pairwise mechanisms, such as reducing the sodium and transient calcium conductances of the PY neuron, can reduce the energy consumption of the entire circuit by up to 55%. When considering only the energy consumed in a specific neuron, pairwise mechanisms can reduce energy consumption by up to 80% (*SI Appendix*, Fig. S22). Finally, pairwise mechanisms between synapses and conductances of the respective postsynaptic neurons can reduce energy consumption of the entire circuit by up to 43%.

These analyses provide hypotheses for causal mechanisms for how neurons can be tuned into low-energy regimes, while the neural activity keeps satisfying functional constraints. We demonstrated that even simple mechanisms involving one or two conductances can have a substantial impact on the energy consumption of the circuit—thus, low-energy configurations can be found with local parameter changes, not requiring fine coordination among multiple parameters.

Neurons Can Be Tuned Individually to Achieve Minimal Circuit Energy. Next, we asked how single neurons interact to produce functional and efficient circuit activity. Can the energy of the entire circuit be minimized by optimizing the energy of each neuron individually? Does the circuit retain functional activity when

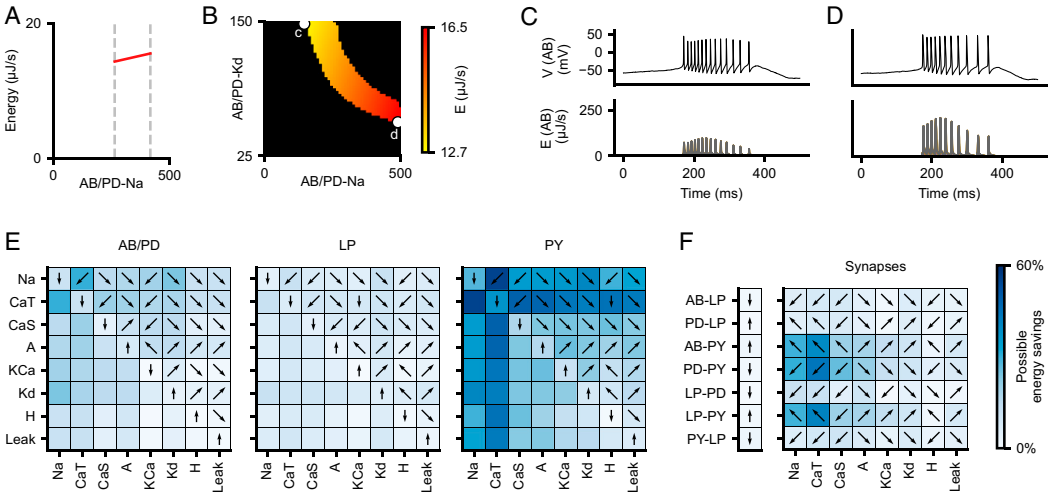


Fig. 5. Minimal tuning mechanisms for low energy consumption. (*A*) Time-averaged energy consumption (as predicted by linear regression) of several models that differ only in their maximal sodium conductance in the AB/PD neuron. Energy increases with $\bar{g}_{Na}$. We excluded circuits with too low and too high values of $\bar{g}_{Na}$, for which the model does not reproduce experimental data. (*B*) Same as *A* but for models that differ in their maximal conductances of sodium (Na) and delayed-rectifier potassium (K_d) in the AB/PD neuron. (*C*) (*Top*) Voltage trace of the AB/PD neuron for the most efficient configuration within the plane shown in *B*. (*Bottom*) Energy consumption during that activity. (*D*) Same as *C* but for the least efficient configuration. (*E*) Fraction of energy that can be saved by modifying a single membrane parameter (diagonal of each matrix) or pairs of membrane parameters (upper and lower diagonal). Color bar is the same as in *F*. Arrows indicate the direction in which (pairs of) parameters should change in order to reduce energy: left/right refers to the parameter on the *x* axis, and top/bottom refers to the parameter on the *y* axis. (*F*) Fraction of energy that can be saved by modifying a synaptic conductance (vector on the left) or the synaptic conductance and one membrane conductance of the respective postsynaptic neuron (matrix on the right).

neurons are individually optimized for low energy efficiency? Within our database of 35,939 model configurations, there is a weak correlation between the energies consumed by pairs of neurons, which suggests that the energy consumption between neurons might be independent from one another (Fig. 6*A*; AB/PD versus LP, correlation coefficient $r = -0.006$, P value $p = 0.23$; LP versus PY, $r = 0.02$, $p = 3 \times 10^{-6}$; AB/PD versus PY, $r = -0.03$, $p = 8 \times 10^{-9}$). We thus investigated whether we could optimize the parameters of each neuron individually for low energy consumption and still retain functional circuit activity. We searched our database of 35,939 model configurations for the single-neuron models with minimal energy consumption individually. We selected the five most efficient single-neuron parameter combinations for each of the neurons and assembled them into 125 (5^3) network configurations. We then identified synaptic conductances that match each of these configurations with Markov chain Monte Carlo (Fig. 6*B*; details in *SI Appendix*, SI Text). Notably, given the already estimated full posterior distribution, this step does not require additional simulations.

For each of the 125 combinations of membrane conductances, we found a set of synaptic conductances for which the network activity closely resembles experimentally measured activity (Fig. 6*C*). The resulting configurations have disparate parameters (Fig. 6*D*) but highly similar network activity. Furthermore, we found that the resulting configurations have similar and very low energy consumption. The energy consumption of these circuits is significantly smaller than that of any of the configurations in our database of 35,939 model configurations (Fig. 6*E*). This demonstrates that optimizing a specific neuron for energy efficiency does not preclude the connected neurons from being energy efficient. Thus, our results suggest that the pyloric network can be optimized for energy efficiency by tuning neurons individually for low energy consumption.

We estimated how likely these energy-efficient circuits are under the estimated posterior. We found that all these models have similar posterior log-probability as the 35,939 model configurations in our database (Fig. 6*F*); i.e., they are part of the inferred posterior and are as likely to underlie the experimentally measured activity as the database models. How, then, are these newly found configurations so much more energy-efficient than configurations sampled from the posterior? Given that the posterior is a high-dimensional distribution (31 dimensions), drawing random samples from such a distribution will unlikely result in configurations for which all three neurons are optimally efficient, and we cannot exclude the possibility that there might be unsampled regions in parameter space with even more energy-efficient circuit configurations.

Robustness to Temperature Does Not Require an Increased Metabolic Cost. The crab *C. borealis* experiences daily and yearly fluctuations in temperature which in turn influence the chemical and physical properties of neurons (32–34). Nonetheless, neural circuits such as the pyloric network can maintain their functionality in the presence of these temperature variations. As temperature increases, the cycle frequency of the circuit increases exponentially, but the phases between bursts remain relatively constant (35, 39). We investigated whether the pyloric network trades off robustness to changes in temperature with energy efficiency, i.e., whether temperature-robust solutions are more energetically costly.

The temperature dependence of a biophysical parameter R is captured by the Q_{10} value and is defined as follows:

$$R_T = R_{ref}\, Q_{10}^{(T - T_{ref})/10},$$

where R_{ref} is the parameter value at the reference temperature $T_{ref} = 11\ ^{\circ}\mathrm{C}$. We extended the model of the pyloric network to include Q_{10} values for all maximal membrane and synaptic

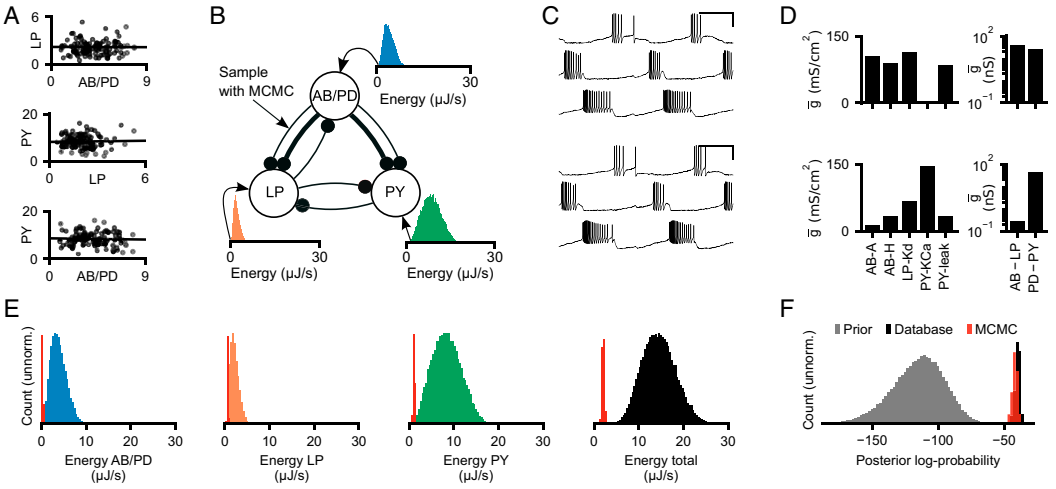


Fig. 6. Neurons can be tuned individually to achieve minimal circuit energy consumption. (*A*) Black dots indicate energy consumed by each neuron separately. Shown are 100 randomly selected parameter configurations from our database of 35,939 configurations. Linear regression (black line) shows a weak correlation between the energy consumed by pairs of neurons. (*B*) We select the five most efficient parameter configurations for each neuron separately and search with Markov chain Monte Carlo for synaptic conductances such that the target circuit activity is achieved. (*C*) The activity produced by two parameter configurations produced with the strategy described in *B*. (*D*) A subset of the (*Left*) membrane and (*Right*) synaptic conductances for the configurations in *C*. Despite generating similar network activity, the configurations have very different circuit parameters. The membrane conductances are scaled with the following factors (left to right): 10, 10,000, 1, 100, and 10,000. (*E*) Histogram over the energy consumption (time averaged) of all 35,939 models in our database (blue, orange, green, and black) and the energy consumption of the configurations produced with the strategy described in *B* (red). (*F*) Histogram of the posterior log-probability for samples from the prior distribution (gray), for the 35,939 models in our database (black), and for the configurations produced with the strategy described in *B* (red).

conductances (details in *SI Appendix*, SI Text) (40, 41). We then used SNPE to identify all maximal membrane and synaptic conductances, as well as the associated Q_{10} values (41 parameters in total) that match experimental recordings at 11 and 27 °C (Fig. 7*A*) (38). We set the previously identified posterior distribution (Fig. 2*E*) over circuit parameters given experimental data at 11 °C as the new prior distribution and then applied SNPE to match the model with experimental data at 27 °C (Fig. 7*B*; full posterior in *SI Appendix*, Fig. S23, and details in *SI Appendix*, SI Text). We sampled circuit parameters and Q_{10} values from the resulting distribution and selected samples whose activity closely matched experimental data at 11 and 27 °C (Fig. 7*C*). Overall, we generated a database of 967 sets of circuit parameters and Q_{10} values. When simulating at temperatures between 11 and 27 °C, these circuits show the characteristic exponential increase in cycle frequency as well as the constant phase relationship between bursts observed experimentally (Fig. 7*D*) (35).

We asked whether the energy consumed by the circuit at 11 °C is proportional to the energy consumed at 27 °C. We found that despite the number of spikes in our model being higher at higher temperatures, the total energy consumption is lower at 27 °C (Fig. 7*E*; note that for one of the three preparations, the energy consumptions at 11 and 27 °C are similar; *SI Appendix*, Fig. S9). This occurs because at higher temperatures, the increase in the number of spikes is accompanied by a decrease in channel time constants and respective decrease in energy per spike (*SI Appendix*, Fig. S24). In addition, there is a clear correlation between energy consumptions at 11 and 27 °C (Pearson correlation coefficient, 0.66), although circuit configurations with similar efficiency at 11 °C can show a range of energy consumptions at 27 °C (Fig. 7*E*).

We then investigated how the additional constraint of temperature robustness impacts the parameter degeneracy of the pyloric network. We computed the SD of the maximal conductances across the models that match experimental data at 11 and 27 °C and whose energy consumption is in the 2% quantile at both temperatures (Fig. 7*F*). The resulting SD is smaller than that of all models in our database of 35,939 models, but a large parameter variability remains. Thus, we found a substantial parameter degeneracy in circuits constrained by "pyloric-ness," energy efficiency, and temperature robustness.

Does temperature robustness have an influence on metabolic cost? We computed the energy consumed at 11 °C for three different scenarios: first, for all models in our database of 35,939 model configurations matching experimental data recorded at 11 °C (same as Fig. 2*H*); second, for all models in our database of 35,939 model configurations that are also functional at 27 °C (i.e., produce triphasic activity); and third, for all models in our database of 967 model configurations matching experimental data recorded at 11 and 27 °C. In all three of these scenarios, the distribution of metabolic cost was similar (Fig. 7*G*; note that the slightly different average energy consumption between the first and the third scenarios occurred only in two of the three preparations; *SI Appendix*, Figs. S6 and S9). In particular, all three scenarios contained configurations that produce energy-efficient circuit function. This demonstrates that enforcing temperature robustness does not require the pyloric network to be less energy efficient.

Overall, our analyses indicate that the model of the pyloric network retains substantial parameter degeneracy despite constraints on energy efficiency and temperature robustness. In addition, we showed that temperature robustness does not entail additional metabolic cost.

Discussion

Neural systems undergo environmental and neuromodulatory perturbations to their mechanisms. The parameter degeneracy

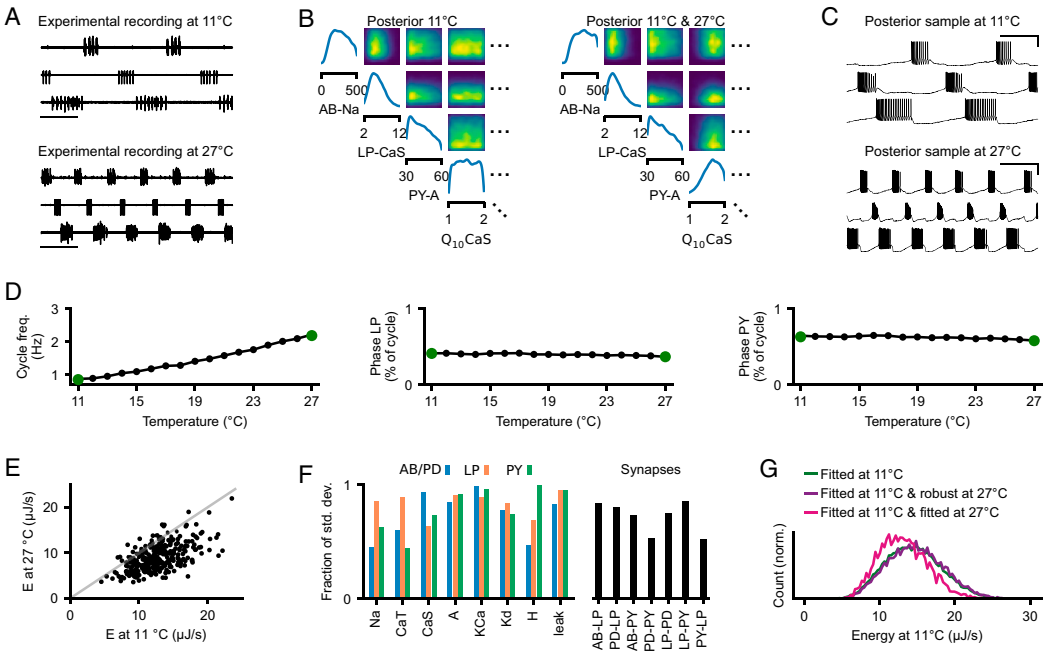


Fig. 7. Temperature robustness does not preclude energy efficiency. (*A*) (*Top*) Experimental data at 11 °C. (*Bottom*) Experimental data at 27 °C (38). (*B*) (*Left*) Posterior distribution given experimental data at 11 °C. (*Right*) Posterior distribution given experimental data at 11 and 27 °C. (*C*) Simulations for a parameter set drawn from the posterior distribution matching experimental data at (*Top*) 11 °C and (*Bottom*) 27 °C. (*D*) (*Left*) Cycle frequency, (*Middle*) phase of LP neuron, and (*Right*) phase of PY neuron for parameter set shown in *C*, simulated at temperatures between 11 and 27 °C. Green dots are the values of the experimental preparations. (*E*) Energy consumption at 11 °C versus 27 °C (time averaged) for 967 circuits sampled from the posterior (in *B*, *Right*). The identity line is shown in gray. (*F*) SD of parameters for models that match experimental data at 11 and 27 °C and that have energy consumption in the lowest 2% quantile at 11 and 27 °C. SD is normalized to the SD of the parameters across all 35,939 models in our database. (*G*) Distribution of the energy consumption of circuits matching experimental data at 11 °C (green), distribution of the energy consumption of circuits that match data at 11 °C and are robust at 27 °C (purple), and distribution of the energy consumption of circuits that match experimental data at 11 and 27 °C (pink).

of neural systems, i.e., the ability to generate similar activity from disparate parameters, confers a certain degree of robustness to such perturbations (7–10, 42, 43). However, not all system configurations might be equally desirable, with some configurations being more energy efficient than others (15). Here we analyzed the energy consumption of parameter configurations with similar activity in the pyloric network of the stomatogastric ganglion. We found that even when the network activity is narrowly tuned to experimental data, the energy consumption can strongly vary between parameter configurations. Despite this diversity of metabolic costs, energy-efficient activity could be produced from a wide range of circuit parameters. When characterizing the range of data-consistent parameters, we found a linear relationship between circuit parameters and energy consumption, which allowed us to identify tuning mechanisms for low energy consumption. Last, we showed that temperature robustness does not preclude energy efficiency and that parameter degeneracy remains despite metabolic and temperature constraints. These findings were facilitated by a methodological advance that increased the efficiency of previously published tools for simulation-based inference (14, 31, 44, 45).

Parameter Degeneracy under Multiple Constraints. In addition to a specific activity, neural circuits are likely constrained by other requirements, e.g., low energy consumption or robustness to perturbations such as fluctuations in temperature or pH (35, 40, 41, 46–50). Here we investigated how energy efficiency impacts the parameter degeneracy of neural systems. While a plausible hypothesis would have been that energy efficiency reduces or eliminates degeneracy altogether, here we found that parameter degeneracy is preserved, even within circuits with very low energy consumption. We identified multiple parameter directions that influence energy consumption only weakly, thus allowing systems to produce energy-efficient network activity with widely disparate circuit parameters. However, we also showed that there are directions in parameter space which drastically impact energy consumption and that changes along these directions lead to configurations which can differ almost by an order of magnitude in their energy consumption. Our results, thus, suggest that changes of most parameters only weakly affect energy consumption, yet some parameters are crucial and should be tuned to achieve low energy consumption.

In our work, parameter degeneracy consisted in the range of pyloric network models that match specific features of experimental activity. We used the same features as in previous work (5), which are physiological constraints of the pyloric network, e.g., cycle duration, burst durations, and gaps and phases of bursts. However, we cannot discard the possibility that the inclusion of additional data features (e.g., spike height or spike width) would have impacted parameter degeneracy and consequently also the range of energies.

Previous work demonstrated that multiple parameter sets in a model of the AB/PD neuron are temperature robust (40).

Here we investigated the interplay between energy consumption and temperature robustness at the circuit level and showed that functional, energy-efficient, and temperature-robust activity can be generated from disparate circuit parameters. In addition, consistent with previous work in a single-neuron model of the grasshopper (28), we found that temperature robustness does not require an increased metabolic cost. Whether these results will generalize with the inclusion of the robustness to additional external perturbations, e.g., pH fluctuations (49, 51), or internal perturbations, e.g., neuromodulation (39), remains a subject for future work.

O'Leary and Marder (48) have demonstrated in a model of the PD neuron that some physiological features (such as duty cycle) can be maintained under temperature perturbations when conductances are scaled by a common factor. We tested the possibility that such invariance under conductance scaling could explain the parameter degeneracy and ranges of energies observed in our circuit model: scaling the conductances by a common factor would scale the currents and thereby the energy consumption. However, for the parameter ranges we used [similar ranges as in Prinz et al. (5)], scaling the conductances changed physiological features (such as the cycle duration) of the pyloric rhythm and led to the model not fitting the experimental data accurately (*SI Appendix*, Fig. S25).

More generally, whether there is potential for a system to exhibit parameter degeneracy depends on the number of constraints on the system relative to the number of free parameters: in an overparameterized system, if there is any parameter setting which satisfies the constraints, it is expected that there will be multiple such settings. Our model has 31 conductances and 10 Q_{10} values, and we use 18 voltage features at 11 °C, one energy consumption constraint, and 18 voltage features at 27°. While there is a similar number of constraints relative to the parameter dimensionality, some of those constraints are likely redundant, in which case we have fewer constraints than parameters. Thus, the fact that there are multiple feasible parameter settings is not surprising per se. However, rather than these multiple solutions corresponding to similar parameter values, we found these to be quite disparate in the parameter space.

Relation to Previous Work on Metabolic Cost of Neural Systems. There has been extensive work on quantifying the metabolic cost of biophysical processes in single neurons (15, 22–26) and how single neurons subject to functional constraints can be tuned to minimize energy consumption (15, 16, 23, 25). Consistent with this work, we found that total energy consumption of the pyloric network is strongly influenced by the sodium current (25) but also by the transient and slow calcium currents. The maximal sodium conductance is the most prominent driver of the energy per spike: increases in the conductance lead to an increase of metabolic cost per spike (15, 25). In contrast, calcium currents influence energy consumption through the number of spikes within a burst, despite not consuming much energy themselves. Our results suggest that the maximal conductances of sodium and calcium might be regulated for metabolic efficiency. We thus predict that these conductances are less variable in nature than expected by computational models only matching network activity. Nevertheless, we should note that our findings are based on two simplifying assumptions: first, we studied simple single-compartment neurons rather than more realistic multicompartment neuron models (52), and second, the energy measure is derived directly from the Hodgkin–Huxley model (36), rather than taking into account all the complexity of the ionic exchange leading to ATP consumption (15, 21, 23, 25).

While our study characterizes which parameter configurations are energy efficient, we did not study the mechanisms of how biological circuits would modulate their conductances in order to arrive at these configurations. Long-term changes in intracellular calcium have been found to regulate many ion channels (53–55); thus, it is possible that intracellular calcium is used to tune conductances for energy efficiency. In our study, we found that intracellular calcium level is linearly related to energy consumption across circuit configurations that match experimental data (*SI Appendix*, Fig. S26). It is, therefore, conceivable that the intracellular calcium concentration is used as a rough estimate of the energy consumption and as a feedback signal to tune membrane and synaptic conductances into energy-efficient regimes, subject to additional constraints such as functional activity and robustness to various perturbations (55).

Previous studies have demonstrated that synaptic mechanisms can consume a substantial amount of energy (21, 56, 57). In contrast, in the considered model of the pyloric network, synaptic currents consume only a minor fraction of energy [∼0.08% of the total energy is consumed by synapses, whereas Attwell and Laughlin (21) report 40% of energy being consumed by synaptic mechanisms]. A potential reason for this difference is the low number of connections in our model of the pyloric network: each model neuron projects to up to three other model neurons, whereas the synaptic energy consumption reported in Attwell and Laughlin (21) is based on the assumption of 8,000 synaptic boutons per neuron. Cabirol-Pol et al. (58) report only few synaptic sites between the PD and LP neuron in the lobster *Homarus gammarus*. Thus, models of more complex neural circuits driven by excitatory, recurrent connectivity, such as the ones found in the cortex, might spend a larger fraction of energy on synaptic mechanisms.

We used a measure of energy consumption that is based on an equivalent electrical circuit. This measure ignores a range of internal processes taking place within biological neurons, e.g., the efficiency of ion pumps or the vesicle release at synapses (21), and, therefore, underestimates the true energy consumption of the biological circuit. We reproduced our main results for a current-based energy measure taking into account the efficiency of pumps (*SI Appendix*, Fig. S3). While the absolute values of the energy consumption changed (see different ranges of energy in *SI Appendix*, Fig. S3 *A* and *D* as compared to Figs. 2*H* and 7*G*), the results are highly similar due to the proportionality between the two measures ($\rho = 0.999$; *SI Appendix*, Figs. S1 and S2). In addition, a more detailed model of synaptic energy consumption (including, e.g., vesicle release) would likely change the cost of synaptic activity, although synaptic cost constitutes only a very small fraction (0.08%) of total energy consumption in our circuit model. It is, thus, unlikely that the use of synapses with more biophysical details would entirely change the conclusions of our study. Additional factors such as glutamate recycling, presynaptic Ca^{2+} entry, and maintaining resting potentials would also modify our estimate of the energy consumption, but these factors are reported to consume a rather small fraction (<20%) of energy expenditure in the brain (21).

Energy Efficiency in the Pyloric Network. Experimental studies have shown that the parameters of the pyloric network vary across wide ranges (1, 2, 59). This raises the question of whether these disparate solutions are all tuned for energy efficiency. In our study, we demonstrated that energy-efficient circuit function can be compatible with many parameter configurations. Therefore, despite the variability of the parameters, each configuration in the crab *C. borealis* might be tuned for low energy consumption.

However, the pyloric network is a small subset of the nervous system of the crab and, therefore, likely consumes a small fraction of its total energy budget. Thus, even if the nervous system of the crab is tuned for energy efficiency, it could still achieve this without strict energy requirements for the pyloric network.

Increasing the Efficiency of Simulation-Based Inference. We used a previously introduced tool, SNPE (14, 45), to identify all models consistent with experimentally measured activity as well as prior knowledge about realistic parameter ranges. We improved the efficiency of SNPE by introducing a classifier that rejects invalid simulations (31). By using this classifier, we were able to improve the accuracy of SNPE while requiring only half as many simulations (14). Because of this larger simulation budget, the resulting posterior distributions became more accurate. Furthermore, the trained neural density estimator is amortized; i.e., one can obtain the posterior distribution for multiple experimental preparations without running further simulations or training a new neural network.

The classifier-enhanced SNPE can be applied to other modeling studies in neuroscience. In particular, the classifier to predict invalid simulations is valuable whenever there are parameter values for which the computational model of interest produces ill-defined features: e.g., the spike shape cannot be defined in cases where a neuron model does not produce spikes. Our method has the potential to significantly speed up inference in these scenarios.

Implications for the Operation of Neural Circuits. Our findings suggest that neural circuits can be energy efficient with largely disparate biophysical parameters, even with highly specific functional requirements under naturally occurring perturbations. This raises the question of whether such energy efficiency is present in real biological systems and how these systems could be tuned for metabolic efficiency.

Materials and Methods

Code to reproduce the figures is available at https://github.com/mackelab/stg_energy. Code for running SNPE and training a classifier to reject invalid simulations is available in our toolbox: https://github.com/mackelab/sbi (60). A tutorial for how to use these features can be found on our website: https://www.mackelab.org/sbi. The simulator of the pyloric network in Cython (61) is available at https://github.com/mackelab/pyloric.

We analyzed extracellular recordings of the stomatogastric motor neurons that are involved in the triphasic pyloric rhythm in the crab *C. borealis* (38, 39). The circuit model of the crustacean stomatogastric ganglion was adapted from Prinz et al. (5). To compute the energy consumption of a specific network activity, we followed the approach of Moujahid et al. (36). In order to find circuit models that are consistent with the neural data, we used an extension of SNPE (14). Further details about the data, modeling, and inference are available in *SI Appendix*, SI Text.

Data, Materials, and Software Availability. Computer code has been deposited in GitHub (https://github.com/mackelab/stg_energy) (62). Previously published data were used for this work (38).

ACKNOWLEDGMENTS. We thank Sara A. Haddad and Eve Marder for sharing their data and discussions; Martin Stemmler for discussions; and Poornima Ramesh, Richard Gao, and Jan Boelts for discussions and comments on the manuscript. We also thank the reviewers for a constructive review process which significantly improved the manuscript. M.D. is supported by the International Max Planck Research School for Intelligent Systems. This work was supported by the German Research Foundation through Sonderforschungsbereich (SFB) 1089 "Synaptic Microcircuits" and Germany's Excellence Strategy–EXC-Number 2064/1, project 390727645, as well as the German Federal Ministry of Education and Research (project ADIMEM, FKZ 01IS18052 A-D and Tübingen AI Center, FKZ 01IS18039A).

Author affiliations: [a]Machine Learning in Science, Excellence Cluster "Machine Learning," Tübingen University, 72076 Tübingen, Germany; [b]Max Planck Institute for Intelligent Systems, Department of Empirical Inference, 72076 Tübingen, Germany; and [c]Max Planck Institute for Neurobiology of Behavior – caesar, 53175 Bonn, Germany

1. J. Golowasch, L. F. Abbott, E. Marder, Activity-dependent regulation of potassium currents in an identified neuron of the stomatogastric ganglion of the crab Cancer borealis. *J. Neurosci.* **19**, RC33 (1999).
2. J. Golowasch, M. S. Goldman, L. F. Abbott, E. Marder, Failure of averaging in the construction of a conductance-based neuron model. *J. Neurophysiol.* **87**, 1129–1131 (2002).
3. M. S. Goldman, J. Golowasch, E. Marder, L. F. Abbott, Global structure, robustness, and modulation of neuronal models. *J. Neurosci.* **21**, 5229–5238 (2001).
4. A. A. Prinz, C. P. Billimoria, E. Marder, Alternative to hand-tuning conductance-based models: Construction and analysis of databases of model neurons. *J. Neurophysiol.* **90**, 3998–4015 (2003).
5. A. A. Prinz, D. Bucher, E. Marder, Similar network activity from disparate circuit parameters. *Nat. Neurosci.* **7**, 1345–1352 (2004).
6. R. C. Roffman, B. J. Norris, R. L. Calabrese, Animal-to-animal variability of connection strength in the leech heartbeat central pattern generator. *J. Neurophysiol.* **107**, 1681–1693 (2012).
7. G. M. Edelman, J. A. Gally, Degeneracy and complexity in biological systems. *Proc. Natl. Acad. Sci. U.S.A.* **98**, 13763–13768 (2001).
8. E. Marder, A. L. Taylor, Multiple models to capture the variability in biological neurons and networks. *Nat. Neurosci.* **14**, 133–138 (2011).
9. J. N. MacLean, Y. Zhang, B. R. Johnson, R. M. Harris-Warrick, Activity-independent homeostasis in rhythmically active neurons. *Neuron* **37**, 109–120 (2003).
10. J. N. MacLean *et al.*, Activity-independent coregulation of IA and Ih in rhythmically active neurons. *J. Neurophysiol.* **94**, 3601–3617 (2005).
11. R. N. Gutenkunst *et al.*, Universally sloppy parameter sensitivities in systems biology models. *PLOS Comput. Biol.* **3**, 1871–1878 (2007).
12. R. Grashow, T. Brookings, E. Marder, Compensation for variable intrinsic neuronal excitability by circuit-synaptic interactions. *J. Neurosci.* **30**, 9145–9156 (2010).
13. T. O'Leary, A. C. Sutton, E. Marder, Computational models in the age of large datasets. *Curr. Opin. Neurobiol.* **32**, 87–94 (2015).
14. P. J. Gonçalves *et al.*, Training deep neural density estimators to identify mechanistic models of neural dynamics. *eLife* **9**, e56261 (2020).
15. A. Hasenstaub, S. Otte, E. Callaway, T. J. Sejnowski, Metabolic cost as a unifying principle governing neuronal biophysics. *Proc. Natl. Acad. Sci. U.S.A.* **107**, 12329–12334 (2010).
16. B. Sengupta, A. A. Faisal, S. B. Laughlin, J. E. Niven, The effect of cell size and channel density on neuronal information encoding and energy efficiency. *J. Cereb. Blood Flow Metab.* **33**, 1465–1473 (2013).
17. B. Sengupta, M. B. Stemmler, Power consumption during neuronal computation. *Proc. IEEE* **102**, 738–750 (2014).
18. J. Astrup, P. M. Sørensen, H. R. Sørensen, Oxygen and glucose consumption related to Na+-K+ transport in canine brain. *Stroke* **12**, 726–730 (1981).
19. J. Astrup, P. M. Sørensen, H. R. Sørensen, Inhibition of cerebral oxygen and glucose consumption in the dog by hypothermia, pentobarbital, and lidocaine. *Anesthesiology* **55**, 263–268 (1981).
20. L. Sokoloff, Energetics of functional activation in neural tissues. *Neurochem. Res.* **24**, 321–329 (1999).
21. D. Attwell, S. B. Laughlin, An energy budget for signaling in the grey matter of the brain. *J. Cereb. Blood Flow Metab.* **21**, 1133–1145 (2001).
22. H. Alle, A. Roth, J. R. Geiger, Energy-efficient action potentials in hippocampal mossy fibers. *Science* **325**, 1405–1408 (2009).
23. M. B. Stemmler, B. Sengupta, S. Laughlin, J. Niven, "Energetically optimal action potentials" in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K. Q. Weinberger, Eds. (Curran Associates, Inc., 2011), pp. 1566–1574.
24. B. C. Carter, B. P. Bean, Sodium entry during action potentials of mammalian neurons: Incomplete inactivation and reduced metabolic efficiency in fast-spiking neurons. *Neuron* **64**, 898–909 (2009).
25. B. Sengupta, M. Stemmler, S. B. Laughlin, J. E. Niven, Action potential energy efficiency varies among neuron types in vertebrates and invertebrates. *PLOS Comput. Biol.* **6**, e1000840 (2010).
26. G. Yi, Y. Fan, J. Wang, Metabolic cost of dendritic Ca^{2+} action potentials in layer 5 pyramidal neurons. *Front. Neurosci.* **13**, 1221 (2019).
27. S. Onasch, J. Gjorgjieva, Circuit stability to perturbations reveals hidden variability in the balance of intrinsic and synaptic conductances. *J. Neurosci.* **40**, 3186–3202 (2020).
28. F. A. Roemschied, M. J. Eberhard, J. H. Schleimer, B. Ronacher, S. Schreiber, Cell-intrinsic mechanisms of temperature compensation in a grasshopper sensory receptor neuron. *eLife* **3**, e02078 (2014).
29. R. M. Harris-Warrick *et al.*, *Dynamic Biological Networks: The Stomatogastric Nervous System* (MIT Press, 1992).
30. E. Marder, D. Bucher, Understanding circuit dynamics using the stomatogastric nervous system of lobsters and crabs. *Annu. Rev. Physiol.* **69**, 291–316 (2007).
31. J. M. Lueckmann *et al.*, "Flexible statistical inference for mechanistic models of neural dynamics" in *Advances in Neural Information Processing Systems*, I. Guyon *et al.*, Eds. (Curran Associates, Inc., 2017), pp. 1289–1299.
32. L. Stehlik, C. MacKenzie Jr., W. Morse, Distribution and abundance of four brachyuran crabs on the northwest Atlantic shelf. *Fish Bull.* **89**, 473–492 (1991).
33. M. J. Donahue *et al.*, Predation risk, prey abundance, and the vertical distribution of three brachyuran crabs on Gulf of Maine shores. *J. Crustac. Biol.* **29**, 523–531 (2009).
34. C. J. Krediet, M. J. Donahue, Growth-mortality trade-offs along a depth gradient in cancer borealis. *J. Exp. Mar. Biol. Ecol.* **373**, 133–139 (2009).
35. L. S. Tang *et al.*, Precise temperature compensation of phase in a rhythmic motor pattern. *PLoS Biol.* **8**, e1000469 (2010).
36. A. Moujahid, A. d'Anjou, F. J. Torrealdea, F. Torrealdea, Energy and information in Hodgkin-Huxley neurons. *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.* **83**, 031912 (2011).

37. L. M. Alonso, E. Marder, Visualization of currents in neural models with similar behavior and different conductance densities. *eLife* **8**, e42722 (2019).
38. S. A. Haddad, E. Marder, Recordings from the *C. borealis* stomatogastric nervous system at different temperatures in the decentralized condition (2021). https://doi.org/10.5281/zenodo.5139650. Accessed 1 August 2021.
39. S. A. Haddad, E. Marder, Circuit robustness to temperature perturbation is altered by neuromodulators. *Neuron* **100**, 609–623.e3 (2018).
40. J. S. Caplan, A. H. Williams, E. Marder, Many parameter sets in a multicompartment model oscillator are robust to temperature perturbations. *J. Neurosci.* **34**, 4963–4975 (2014).
41. L. M. Alonso, E. Marder, Temperature compensation in a small rhythmic circuit. *eLife* **9**, e55470 (2020).
42. E. Marder, Variability, compensation, and modulation in neurons and circuits. *Proc. Natl. Acad. Sci. U.S.A.* **108** (suppl. 3), 15542–15548 (2011).
43. E. Marder, M. L. Goeritz, A. G. Otopalik, Robust circuit rhythms in small circuits arise from variable circuit components and mechanisms. *Curr. Opin. Neurobiol.* **31**, 156–163 (2015).
44. G. Papamakarios, T. Pavlakou, I. Murray, "Masked autoregressive flow for density estimation" in *Advances in Neural Information Processing Systems*, I. Guyon *et al.*, Eds. (Curran Associates, Inc., 2017), pp. 2338–2347.
45. D. Greenberg, M. Nonnenmacher, J. Macke, "Automatic posterior transformation for likelihood-free inference" in *International Conference on Machine Learning*, K. Chaudhuri, R. Salakhutdinov, Eds. (PMLR, 2019), pp. 2404–2414.
46. L. S. Tang, A. L. Taylor, A. Rinberg, E. Marder, Robustness of a rhythmic circuit to short- and long-term temperature changes. *J. Neurosci.* **32**, 10075–10085 (2012).
47. A. Rinberg, A. L. Taylor, E. Marder, The effects of temperature on the stability of a neuronal oscillator. *PLOS Comput. Biol.* **9**, e1002857 (2013).
48. T. O'Leary, E. Marder, Temperature-robust neural function from activity-dependent ion channel regulation. *Curr. Biol.* **26**, 2935–2941 (2016).
49. J. A. Haley, D. Hampton, E. Marder, Two central pattern generators from the crab, *Cancer borealis*, respond robustly and differentially to extreme extracellular pH. *eLife* **7**, e41877 (2018).
50. S. Gorur-Shandilya *et al.*, Mapping circuit dynamics during function and dysfunction. *eLife* **11**, e76579 (2022).
51. J. Ratliff, A. Franci, E. Marder, T. O'Leary, Neuronal oscillator robustness to multiple global perturbations. *Biophys. J.* **120**, 1454–1468 (2021).
52. G. Le Masson, S. Przedborski, L. F. Abbott, A computational model of motor neuron degeneration. *Neuron* **83**, 975–988 (2014).
53. M. E. Barish, Intracellular calcium regulation of channel and receptor expression in the plasmalemma: Potential sites of sensitivity along the pathways linking transcription, translation, and insertion. *J. Neurobiol.* **37**, 146–157 (1998).
54. T. O'Leary, M. C. van Rossum, D. J. Wyllie, Homeostasis of intrinsic excitability in hippocampal neurones: Dynamics and mechanism of the response to chronic depolarization. *J. Physiol.* **588**, 157–170 (2010).
55. T. O'Leary, A. H. Williams, A. Franci, E. Marder, Cell types, network homeostasis, and pathological compensation from a biologically plausible ion channel expression model. *Neuron* **82**, 809–821 (2014).
56. W. B. Levy, R. A. Baxter, Energy-efficient neuronal computation via quantal synaptic failures. *J. Neurosci.* **22**, 4746–4755 (2002).
57. B. Sengupta, M. B. Stemmler, K. J. Friston, Information and efficiency in the nervous system–A synthesis. *PLOS Comput. Biol.* **9**, e1003157 (2013).
58. M. J. Cabirol-Pol, D. Combes, V. S. Fénelon, J. Simmers, P. Meyrand, Rare and spatially segregated release sites mediate a synaptic interaction between two identified network neurons. *J. Neurobiol.* **50**, 150–163 (2002).
59. D. J. Schulz, J. M. Goaillard, E. Marder, Variable channel expression in identified single and electrically coupled neurons in different animals. *Nat. Neurosci.* **9**, 356–362 (2006).
60. A. Tejero-Cantero *et al.*, sbi: A toolkit for simulation-based inference. *J. Open Source Softw.* **5**, 2505 (2020).
61. S. Behnel *et al.*, Cython: The best of both worlds. *Comput. Sci. Eng.* **13**, 31–39 (2010).
62. M. Deistler, J. H. Macke, P. J. Goncalves, Code for "Energy-efficient network activity from disparate circuit parameters." GitHub. https://github.com/mackelab/stg_energy. Deposited 30 August 2020.

Supporting Information for

Energy efficient network activity from disparate circuit parameters

Michael Deistler, Jakob H. Macke and Pedro J. Gonçalves

Corresponding Author name.
E-mail: jakob.macke@uni-tuebingen.de, pedro.goncalves@uni-tuebingen.de

This PDF file includes:

Supporting Information Text

Data from the crustacean stomatogastric ganglion. We analyzed extracellular recordings of the stomatogastric motor neurons that are involved in the triphasic pyloric rhythm in the crab *Cancer borealis* (1). The first dataset as seen in Fig. 2 and Fig. 7 is from files 845_082_0044 and 845_082_0064, preparation 1. The second dataset as seen in SI Fig. S4 and SI Fig. S6 is from files 857_016_0049 and 857_016_0069, preparation 1. The third dataset as seen in SI Fig. S7 and SI Fig. S9 is from files 845_078_0027 and 845_078_0040, preparation 2. All preparations were decentralized, i.e., the axons of the descending modulatory inputs were severed. The data were recorded at 11°C and 27°C. Full experimental details in Haddad et al. (2).

Circuit model of the crustacean stomatogastric ganglion. The circuit model of the crustacean stomatogastric ganglion was adapted from Prinz et al. (3). The model is composed of three single-compartment neurons, AB/PD, LP, and PY, where the electrically coupled AB and PD neurons are modeled as a single neuron. Each of the model neurons contains 8 currents, a Na^+ current I_{Na}, a fast and a slow transient Ca^{2+} current I_{CaT} and I_{CaS}, a transient K^+ current I_A, a Ca^{2+}-dependent K^+ current I_{KCa}, a delayed rectifier K^+ current I_{Kd}, a hyperpolarization-activated inward current I_H, and a leak current I_{leak}. In addition, the model contains 7 synapses. As in Prinz et al. (3), these synapses are simulated using a standard model of synaptic dynamics (4). The synaptic input current into the neurons is given by $I_s = \overline{g}_s s(V_{post} - V_s)$, where $\overline{g}_s$ is the maximal synapse conductance, V_{post} the membrane potential of the postsynaptic neuron, and V_s the reversal potential of the synapse. The dynamics of the activation variable s are given by

$$\frac{ds}{dt} = \frac{\overline{s}(V_{pre}) - s}{\tau_s},$$

with

$$\overline{s}(V_{pre}) = \frac{1}{1 + \exp((V_{th} - V_{pre})/\delta)} \quad \text{and} \quad \tau_s = \frac{1 - \overline{s}(V_{pre})}{k_-}.$$

Here, V_{pre} is the membrane potential of the presynaptic neuron, V_{th} is the half-activation voltage of the synapse, δ sets the slope of the activation curve, and k_- is the rate constant for transmitter-receptor dissociation rate.

As in Prinz et al. (3), we model two types of synapses, since AB, LP, and PY are glutamatergic neurons whereas PD is cholinergic. For simplicity and given that varying the synaptic parameters (except synaptic conductances) does not influence energy consumption as strongly as varying maximal conductances (SI Fig. S27), we kept these synaptic parameters fixed throughout this study. We set $E_s = -70$ mV and $k_- = 1/40$ ms for all glutamatergic synapses and $E_s = -80$ mV and $k_- = 1/100$ ms for all cholinergic synapses. For both synapse types, we set $V_{th} = -35$ mV and $\delta = 5$ mV. The membrane area is $0.628 \cdot 10^{-3}$ cm^2.

For each set of membrane and synaptic conductances, we numerically simulate the circuit for 10 seconds with a step size of 0.025 ms. We assumed a simple aggregate model of current noise, by adding a small Gaussian noise current with mean zero and standard deviation 0.001 μA at each time step. This approach could be extended by using more realistic noise models which explicitly account for, e.g., channel noise, stochastic vesicle release, or stochastic calcium dynamics (5). We set the amplitude of the intrinsic noise such that the model roughly matched the variability of the cycle duration across bursts of the experimental traces, based on initial simulations. We did not systematically vary the noise level or noise source, but our Bayesian inference algorithm and analysis pipeline could readily be applied to investigate more detailed noise models in future work, and infer the corresponding parameters. In our current setting, replacing current noise with observation noise did not drastically affect the posterior marginals or disparity of energy consumptions (SI Fig. S12). Nonetheless, we cannot exclude the possibility that using other noise sources such as subunit noise or conductance noise will have a stronger effect on our results.

We applied SNPE to infer the posterior over 24 membrane parameters and 7 synaptic parameters, i.e., 31 parameters in total. The 7 synaptic parameters are the maximal conductances $\overline{g}_s$ of all synapses in the circuit, each of which is varied uniformly in logarithmic domain from 0.01 nS to 1000 nS, with the exception of the synapse from AB to LP, which is varied uniformly in logarithmic domain from 0.01 nS to 10000 nS. The membrane parameters are the maximal membrane conductances for each neuron. The membrane conductances are varied over an extended range of previously reported values (3, 6): The prior distribution over the parameters [Na, CaT, CaS, A, KCa, Kd, H, leak] is uniform with lower bounds $p_{low} = [0, 0, 0, 0, 0, 25, 0, 0]$ mS cm^{-2} and upper bounds $p_{high} = [500, 7.5, 8, 60, 15, 150, 0.2, 0.01]$ mS cm^{-2} for the maximal membrane conductances of the AB neuron, $p_{low} = [0, 0, 2, 10, 0, 0, 0, 0.01]$ mS cm^{-2} and $p_{high} = [200, 2.5, 12, 60, 10, 125, 0.06, 0.04]$ mS cm^{-2} for the maximal membrane conductances of the LP neuron, and $p_{low} = [0, 0, 0, 30, 0, 50, 0, 0]$ mS cm^{-2} and $p_{high} = [600, 12.5, 4, 60, 5, 150, 0.06, 0.04]$ mS cm^{-2} for the maximal membrane conductances of the PY neuron.

We computed 15 summary features proposed by Prinz et al. (3), and 3 additional features (6). The features proposed by Prinz et al. (3) are 15 salient features of the pyloric rhythm, namely: Cycle period T (s), AB/PD burst duration d^b_{AB} (s), LP burst duration d^b_{LP} (s), PY burst duration d^b_{PY} (s), gap AB/PD end to LP start $\Delta t^{es}_{AB\text{-}LP}$ (s), gap LP end to PY start $\Delta t^{es}_{LP\text{-}PY}$ (s), delay AB/PD start to LP start $\Delta t^{ss}_{AB\text{-}LP}$ (s), delay LP start to PY start $\Delta t^{ss}_{LP\text{-}PY}$ (s), AB/PD duty cycle d_{AB}, LP duty cycle d_{LP}, PY duty cycle d_{PY}, phase gap AB/PD end to LP start $\Delta\phi_{AB\text{-}LP}$, phase gap LP end to PY start $\Delta\phi_{LP\text{-}PY}$, LP start phase ϕ_{LP}, and PY start phase ϕ_{PY}. Note that several of these values are only defined if each neuron produces rhythmic bursting behavior. In addition, for each of the three neurons, we computed the maximal duration of its voltage being above -30 mV. We did this as we observed—for many model simulations and in contrast with experimental data—long plateaus at around -10 mV during the bursts, and wanted to detect such traces. If the maximal duration was below 5 ms, we set this

feature to 5 ms. To extract the summary features from the observed experimental data, we first found spikes by searching for local maxima above a hand-picked voltage threshold, and then extracted the 15 above described features. For the experimental preparation, we set the additional 3 features to 5 ms.

At temperatures higher than 11°C, we include Q_{10} values to simulate the biochemical changes of the network parameters. These are defined by an Arrhenius-type factor

$$R_T = R_{ref} Q_{10}^{(T-T_{ref})/10}, \quad [1]$$

where R_{ref} is the parameter value at the reference temperature $T_{ref} = 11$°C, and R_T is the parameter value at temperature T. Each maximal conductance has a different Q_{10}, but the Q_{10} value is the same across neurons (7). We introduce one Q_{10} for the maximal conductances of glutamatergic synapses and one for the cholinergic synapses. The prior distribution for the Q_{10} values is a uniform distribution between 1 and 2 for all maximal conductances (8) but the hyperpolarization-activated inward current, for which the prior bounds are 1 and 4 (9). Since we keep all time constants of the circuit fixed, we also fix the respective Q_{10} values. The Q_{10} values for the time constants are fixed to 2.4 for most m-gates and 2.8 for all h-gates. Following the results from Caplan et al. (10), the Q_{10} values for the time constants of m-gates of KCa and CaS as well as for the time constant of the calcium buffer have lower values: 2.0 for CaS and the calcium buffer and 1.6 for KCa. The Q_{10} value for the time constants of the synapses is 1.7 (within the range of values used in Alonso et al. (7)). Finally, the reversal potential of calcium depends on temperature through the Nernst equation.

Energy consumption. To compute the energy consumption E of a specific network activity, we followed the approach of Moujahid et al. (11). For each neuron, we computed the energy as:

$$E = \int \sum_m g_m (V - V_m)^2 + \sum_s g_s (V - V_s)^2 dt, \quad [2]$$

where g_m is the effective conductance of channel m (i.e., the product of the respective gating variables, maximal conductance and membrane area) and g_s is the effective synaptic conductance s into the specific neuron. V_m is the reversal potential of the membrane current m and V_s is the reversal potential of the synapse s. The units of energy are $[S \cdot V^2 \cdot s = J]$, where S are Siemens, V are Volt, s are seconds, and J are Joules. The total energy consumption was defined as the sum of the energy consumed by each of the three neurons. Throughout this study, we reported the energy per second, which we obtained by dividing the total energy consumption by the duration of the simulation (10 seconds). In addition, as in Moujahid et al. (11), all energy consumptions were normalized by the membrane area of the neurons $0.628 \cdot 10^{-3}$ cm^2. The energy per spike was defined as the energy consumed during bursts divided by the respective number of spikes.

We compared the above power-based measure of energy consumption with a current-based one (12, 13) (SI Fig. S1). To compute this measure, we integrated the sodium and calcium currents over time to obtain the transferred charge:

$$\text{Charge}_m = \int |g_m (V - V_m)| dt, \quad [3]$$

with $m \in \{\text{Na, CaT, CaS}\}$. We then obtained the number of transferred electrons N_m:

$$N_m = \frac{\text{Charge}_m}{e}, \quad [4]$$

where $e \approx 1.6 \cdot 10^{-19} C$ is the elementary charge. The number of hydrolysed ATP molecules for each ion current was obtained by dividing N_m by the valence of the ion and by the number of molecules that are transported with one molecule of ATP (3 for sodium, 2 for calcium):

$$\text{ATP}_m = \frac{N_m}{\text{valence}_m \cdot (\text{molecules per ATP})_m} \quad [5]$$

We estimated the energy consumption in Joule by multiplying the number of ATP molecules by the energy released during the hydrolysis of one ATP molecule, which is approximately $10^{-19} J$ (14). Finally, we divided by the length of the simulation (10 seconds) to obtain the energy in J/s.

In SI Fig. S1, we demonstrate that the power-based and current-based measures correlate very closely. In principle, the two measures are not mathematically proportional: the terms corresponding to a particular current differ by a factor $(V(t) - E_i)$. In practice, given that our models are constrained to have similar voltage waveforms, the factor $(V(t) - E_i)$ is close to a constant across models, and thus the two energy measures are close to proportional. In SI Fig. S2, we demonstrate that slight variations of the voltage waveform only weakly affect the proportionality between these energy measures.

Simulation-based inference. We extended Sequential Neural Posterior Estimation (6) by using a classifier to predict 'invalid' simulation outputs. The resulting algorithm is described in Algorithm 1.

Input: simulator with (implicit) density $p(\mathbf{x}|\boldsymbol{\theta})$, observed data $\mathbf{x}_o$, prior $p(\boldsymbol{\theta})$, rejection criterion $g(\mathbf{x}) \in \{0, 1\}$, classification neural network $G_\zeta(\theta)$, density family q_ψ, neural network $F(\mathbf{x}, \phi)$, number of cycles C of classifier training, simulation count for each cycle N_c

randomly initialize ϕ, ζ
$\tilde{p}_1(\boldsymbol{\theta}) := p(\boldsymbol{\theta})$
$N := 0$
for $c = 1$ ***to*** C **do**
  for $i = 1 \dots N_c$ **do**
    sample $\boldsymbol{\theta}_{N+i} \sim \tilde{p}_c(\boldsymbol{\theta})$
    simulate $\mathbf{x}_{N+i} \sim p(\mathbf{x}|\boldsymbol{\theta}_{N+i})$
    evaluate whether 'valid' $\mathbf{r}_{N+i} = g(\mathbf{x}_{N+i})$
  end
  $N \leftarrow N + N_c$
  train $\zeta \leftarrow \arg\min_\zeta \sum_{j=1}^{N} \mathcal{L}_e(\boldsymbol{\theta}_j, \mathbf{r}_j)$ `// classifier training`
  $T \leftarrow$ Tune classifier threshold s.t. false negative rate $< 1\%$
  $U(\theta) \leftarrow G_\zeta(\theta) > T$
  $\tilde{p}_c(\boldsymbol{\theta}) :\propto U(\theta)p(\boldsymbol{\theta})$
end
train $\phi \leftarrow \arg\min_\phi \sum_{j=1}^{N} \mathcal{L}_d(\boldsymbol{\theta}_j, \mathbf{x}_j)$ `// train neural density estimator`
return $q_{F(\mathbf{x}_o,\phi)}(\boldsymbol{\theta})$

Algorithm 1: SNPE with classifier

Proof of convergence of SNPE with classifier. Below, we prove that the posterior distribution inferred by our method converges to the true posterior distribution. SNPE—with the classifier—minimizes the following loss function with respect to the neural network parameters ϕ:

$$\begin{aligned}\mathcal{L}_d &= -\frac{1}{N}\sum_i \log(q_\phi(\boldsymbol{\theta}_i|\mathbf{x}_i)) \\ &\xrightarrow{N\to\infty} -\mathbb{E}_{p(\boldsymbol{\theta},\mathbf{x})}[\log(q_\phi(\boldsymbol{\theta}|\mathbf{x}))] \\ &= -\mathbb{E}_{U(\boldsymbol{\theta})p(\boldsymbol{\theta})p(\mathbf{x}|\boldsymbol{\theta})}[\log(q_\phi(\boldsymbol{\theta}|\mathbf{x}))],\end{aligned}$$

where $U(\boldsymbol{\theta})$ is a constant $U(\boldsymbol{\theta}) = c > 0$ at least on the posterior support and $U(\boldsymbol{\theta}) = 0$ elsewhere. Then:

$$\begin{aligned}\mathcal{L}_d &= -\iint U(\boldsymbol{\theta})p(\boldsymbol{\theta})p(\mathbf{x}|\boldsymbol{\theta}) \log(q_\phi(\boldsymbol{\theta}|\mathbf{x}))\, d\boldsymbol{\theta} d\mathbf{x} \\ &= -\int p(\mathbf{x}) \int U(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x}) \log(q_\phi(\boldsymbol{\theta}|\mathbf{x}))\, d\boldsymbol{\theta} d\mathbf{x}\end{aligned}$$

Since the integrand of the integral over $\boldsymbol{\theta}$ is proportional to the Kullback-Leibler-divergence between $U(\theta)p(\boldsymbol{\theta}|\mathbf{x})/Z$ and the inferred posterior $q_\phi(\boldsymbol{\theta}|\mathbf{x})$, $\mathcal{L}_d$ is minimized if and only if $q_\phi(\boldsymbol{\theta}|\mathbf{x}) = U(\theta)p(\boldsymbol{\theta}|\mathbf{x})/Z$ for all $\mathbf{x}$ on the support of $p(\mathbf{x})$. Since $U(\theta) > 0$ for all $\boldsymbol{\theta}$ that can generate valid simulations, we obtain $q_\phi(\boldsymbol{\theta}|\mathbf{x}) = p(\boldsymbol{\theta}|\mathbf{x})$ for all valid $\mathbf{x}$.

The proof above ensures that the approximate posterior converges to the true posterior distribution if the training dataset is infinitely large, the density estimator is sufficiently flexible, and the optimization finds the global minimum. While we cannot guarantee these conditions in practice, the posterior distribution obtained is similar to that obtained by previous methods (6).

Classifier of 'valid' simulations. The algorithm includes a classifier $U(\boldsymbol{\theta})$ trained to predict 'valid' simulations. We use a cross-entropy loss $\mathcal{L}_e$. We enforce the classifier $U(\boldsymbol{\theta})$ to be constant, with $U(\boldsymbol{\theta}) = c > 0$ at least on the posterior support and $U(\boldsymbol{\theta}) = 0$ elsewhere:

1. In order for $U(\boldsymbol{\theta})$ to be uniform, we parameterize it as a thresholded binary classifier.
2. To ensure that $c > 0$ at least on the posterior support, we choose the classifier threshold such that there are few false-negatives, i.e., the classifier accepts the parameters if these lie within the posterior support.

If we train the classifier $U(\boldsymbol{\theta})$ with a large enough number of simulations, so that some are 'valid', the trained classifier includes the posterior support. In order to sample from $U(\theta)p(\theta)$, we sample from the prior over parameters $p(\theta)$ and accept the sampled parameters according to the classifier output.

Inference of the posterior distribution given experimental data at 11 °C. Overall, we performed three cycles of simulation and classifier training in order to learn the restricted prior. In the first round, we simulated 3 million parameter sets sampled from the prior. Among these, only 0.97% produced 'valid' summary features. We trained a classifier to detect parameter sets leading to 'valid' simulation outputs. We used a residual neural network with 80 hidden units, two blocks, a dropout rate of 43%, and a batchsize of 199. To deal with 'valid'/'invalid' unbalanced data, we subsampled 'invalid' samples in every epoch. We post-hoc tuned the threshold of the classifier such that the ratio of false-negatives was below 1% on a held-out test set. We then drew 3 million samples from the resulting restricted prior. Out of these, 5.17% produced 'valid' summary features. We then repeated this procedure and out of 3 million simulations from the resulting restricted prior, 8.45% produced good simulations. Overall, in comparison to Gonçalves et al. (6), we used half as many simulations (9 million versus 18.5 million), but generated a database of 'valid' simulations 2.5 times larger. We then used all 438,608 'valid' parameter sets to obtain the posterior distribution with SNPE (see Gonçalves et al. (6) for details). As deep neural density estimator, we chose a neural spline flow (NSF) (15) with 10 transform layers, each consisting of a residual block with two hidden layers, each with 200 hidden units.

Lastly, to ensure that the activity produced by samples from the posterior closely matched experimental data, we sampled 1 million parameter sets from the inferred posterior distribution and performed an additional rejection step, whereby posterior samples had to produce activity within a prescribed distance to the experimental data:

- cycle duration and burst durations deviated from the experimental features by a maximum distance of 0.02 standard deviations of all simulations accepted by the classifier, i.e., 20.6 ms for the cycle duration, and [15.0, 13.5, 11.5] ms for the burst durations (of AB/PD, LP, and PY neurons).
- duty cycles, phase gaps, phase delays, and phases deviated from the experimental features by a maximum distance of 0.2 standard deviations.

Out of 1 million samples from the posterior, 35,939 samples fulfilled all these criteria. Notably, these samples are no longer unbiased samples from the posterior distribution as estimated by SNPE, but they make up a database of model configurations whose activity closely matches experimental data.

Linear regression from circuit parameters onto energy consumption. We performed a linear regression to identify the contribution of the circuit parameters to the total energy consumption using scikit-learn (16). We z-scored the conductances by the mean and standard deviation of the prior. Since the prior bounds of membrane conductances (and thus their standard deviation) could be different between neurons, we chose the maximal standard deviation across the three neurons to z-score the parameters. This ensured that the influence of a particular membrane conductance was comparable between neurons. We z-scored the energy consumption by the mean and standard deviation of configurations within our database of 35,939 samples. Finally, we normalized the linear regression weights w to unit-length.

Regression neural network. In order to test the robustness of the linear regression findings, we trained a regression network to identify directions in the parameter space predictive of total energy consumption. The regression network had the following characteristics: A Residual Network (ResNet) with one hidden layer with 20 hidden units, ReLU activation functions, and 50% dropout rate (17, 18). We trained the network with a mean-squared error loss.

After training the regression network, we searched for directions that were most predictive of the network output $f(\cdot)$. To do so, we followed the procedure described in Constantine et al. (19) and computed:

$$M = \mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta}|\mathbf{x}_o)}[\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})^T]. \quad [6]$$

Intuitively, M captures how much the regression function $f(\cdot)$ changes in different directions of the parameter space, computed as an expected value over posterior samples. We estimated this expected value with a Monte Carlo mean over 10,000 samples from the posterior distribution. We then computed the eigenvalue decomposition of M: The eigenvectors of highest eigenvalue are directions in the parameter space along which the output of the regression neural network is most sensitive to changes.

Minimal tuning mechanisms for low energy consumption. In order to find tuning mechanisms for low energy consumption, we started by varying single or pairs of parameters on a grid while keeping all other parameters constant (at a value taken from our database of 35,939 models). For the matrices in Fig. 5e,f, we repeated the analysis with ten of such models and reported the average.

To predict the energy consumption for any parameter configuration, we used the linear regression model from circuit parameters onto energy consumption described above. To evaluate whether a parameter set matches experimental data, we computed its posterior probability $p(\boldsymbol{\theta}|\mathbf{x}_o)$. If the posterior probability was above a certain threshold t, we considered the parameter set as matching the experimental data, otherwise we discarded the parameter set. To obtain the threshold t, we evaluated the posterior probability of all 35,939 models in our database and used the 10% quantile of these probabilities as t: within our database of models, 90% of the models would have been considered as matching experimental data, whereas 10% of the models would have been (wrongly) discarded. We evaluated different choices for the quantile used to obtain t and obtained qualitatively similar results.

In order to obtain the directions in the parameter space along which energy consumption can be maximally reduced (arrows in Fig. 5e,f), we searched the (1-dimensional or 2-dimensional) grids for the configurations with lowest and highest energy consumption (subject to posterior probability $\geq t$). For each grid, we reported the sign of the direction between these two points of highest and lowest energy consumption.

Sampling synaptic conductances, given energy efficient single-neuron configurations. In order to investigate whether efficient single-neuron parameters could lead to efficient and robust network activity, we first searched our database of 35,939 network configurations for the five configurations that had the lowest metabolic cost in each neuron individually. We combined these single neuron configurations to generate $5^3 = 125$ configurations of membrane conductances. For each of the configurations, we then sampled 1,000 synaptic configurations from the distribution:

$$p(\boldsymbol{\theta}_s|\boldsymbol{\theta}_m, \mathbf{x}_o) \propto p(\boldsymbol{\theta}_s, \boldsymbol{\theta}_m|\mathbf{x}_o), \quad [7]$$

where $\boldsymbol{\theta}_s$ and $\boldsymbol{\theta}_m$ are the synaptic and membrane conductances, respectively. We drew these samples with Markov chain Monte Carlo: Specifically, we used Slice Sampling with axis-aligned updates (20). We then simulated each of these $5^3 \cdot 1000$ configurations. 72 out of the 5^3 configurations contained at least one sample that fulfilled our (distance to experimental data) criteria, and 123 configurations contained a sample that fulfilled a slightly wider criteria (allowing twice as much distance from the experimental data). For each of the other two experimental preparations, we drew another 10,000 samples with MCMC and for each of them found at least one configuration whose activity fulfilled the slightly wider criteria. The histograms in Fig. 6e,f are produced with all simulations that fulfilled the narrow criteria.

Posterior distribution given experimental data at 27°C. In order to infer the posterior distribution given experimental data at 27°C, we started by sampling 3 million parameter sets from the 31-dimensional posterior distribution at 11°C:

$$p(\boldsymbol{\theta}|\mathbf{x}_o^{11}) \propto p(\mathbf{x}_o^{11}|\boldsymbol{\theta})p(\boldsymbol{\theta}).$$

We then drew 3 million sets of Q_{10} values from the prior distribution over Q_{10} values (Q_{10} prior in Methods). We simulated these 3 million parameter sets at 27°C, from which approximately 18% were 'valid' and were used to train a deep neural density estimator (see Methods). The hyperparameters of the neural density estimator were the same as the ones chosen for the inference at 11°C. Since this density estimator was trained on parameters sampled from the posterior distribution at 11°C, the inferred posterior is an approximation to:

$$p(\boldsymbol{\theta}|\mathbf{x}_o^{27}, \mathbf{x}_o^{11}) \propto p(\mathbf{x}_o^{27}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x}_o^{11}) \propto p(\mathbf{x}_o^{27}|\boldsymbol{\theta})p(\mathbf{x}_o^{11}|\boldsymbol{\theta})p(\boldsymbol{\theta}),$$

where $\mathbf{x}_o^{11}$ and $\mathbf{x}_o^{27}$ are the features of the experimental data recorded at 11°C and 27°C, respectively. In other words, the resulting posterior distribution matches prior knowledge about circuit parameters as well as experimental data at 11°C and 27°C. Note that we inferred the posterior distribution at 11°C while ignoring the Q_{10} values because the Q_{10} values, by definition, do not influence the circuit activity at the reference temperature (which is assumed to be 11°C).

Metabolic efficiency at 27°C. For panels Fig. 7e,f and Fig. 7g (pink plot), we analyzed 967 simulations that closely matched experimental data recorded at 11°C and 27°C. For Fig. 7g (purple plot), we simulated, at 27°C, the 35,939 circuit configurations that match experimental data recorded at 11°C. Out of these, 8121 were robust, i.e., displayed pyloric activity at 27°C.

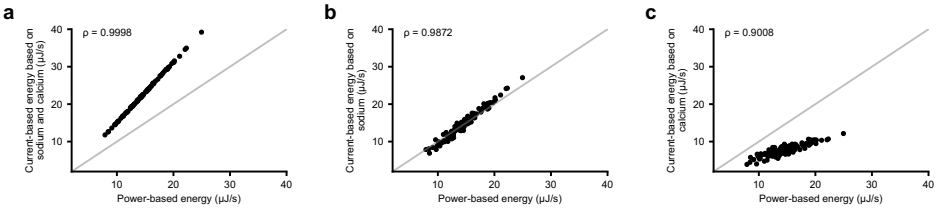
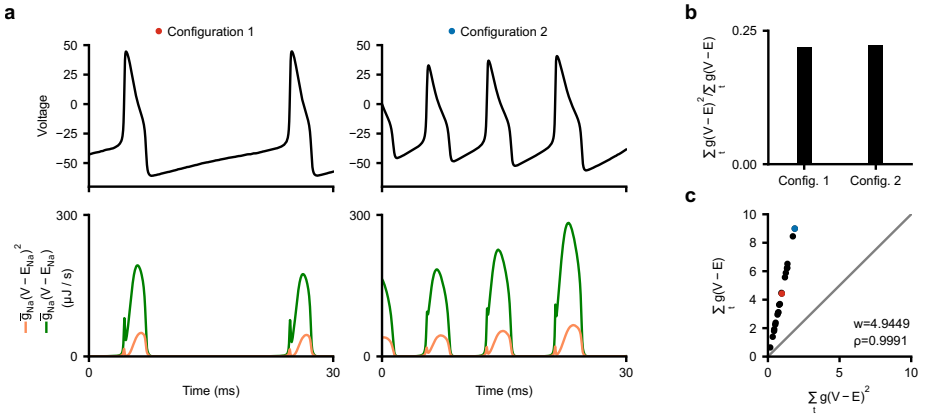


Fig. S1. Comparison between power-based and current-based energy measures. Current-based measures versus power-based measure (11), and respective Pearson correlation coefficients. In (a-c), the power-based measure is computed based on all currents in the model. The current-based measure is computed as a sum of different subsets of currents: (a) Current-based measure based on the sodium, slow calcium, and transient calcium currents. (b) Current-based measure based on the sodium current. (c) Current-based measure based on the slow calcium and transient calcium currents.



194

Fig. S2. Different spike shapes weakly affect the proportionality between energy measures. (a) Top: two voltage traces of the AB/PD neuron from different circuit configurations. Bottom: energy consumption—based only on sodium in the AB/PD neuron—of these traces as measured by the power-based measure (orange) and the current-based measure (green). The current-based measure is converted to μJ/s as described in Methods. (b) Ratio between the two time-averaged energy measures for both configurations. Bar heights: 0.21836, 0.22197. (c) The two energy measures plotted against each other, along with the values for 18 additional parameter configurations. The energy measures are strongly correlated (Pearson correlation coefficient $\rho = 0.9991$) and have a proportionality constant of $w = 4.9449$.

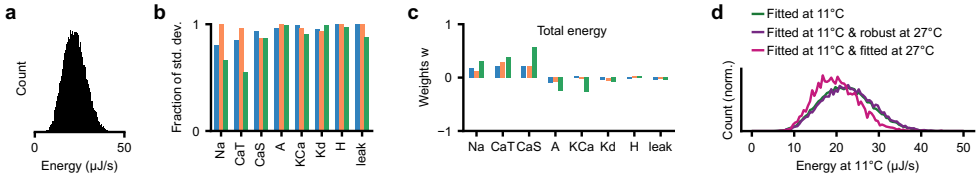
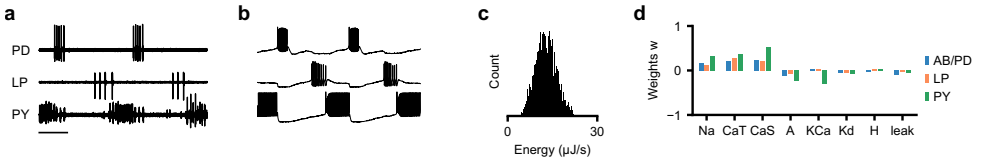
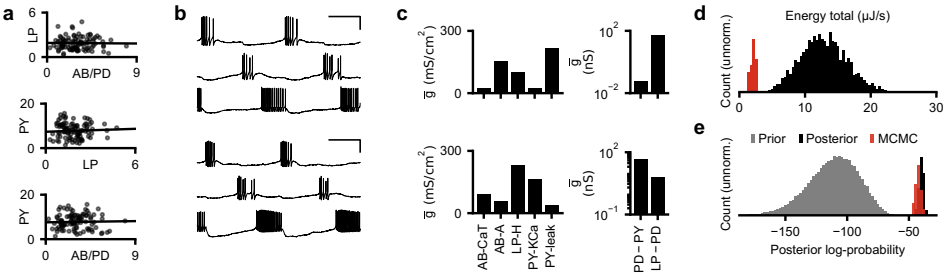


Fig. S3. Reproduction of main results with current-based energy measure. (a) Distribution of current-based energy consumption of all models in our database of $39,939$ model configuration. (b) Standard deviation of parameters for models with current-based energy consumption in the lowest 2% quantile. Standard deviation is normalized to the standard deviation of the parameters across all $35,939$ models in our database. (c) Linear regression weights when predicting current-based energy consumption from circuit parameters. (d) Green: Distribution of the current-based energy consumption of circuits matching experimental data at 11°C. Purple: Distribution of the energy consumption of circuits that match data at 11°C and are robust at 27°C. Pink: Distribution of the energy consumption of circuits that match experimental data at 11°C and 27°C.



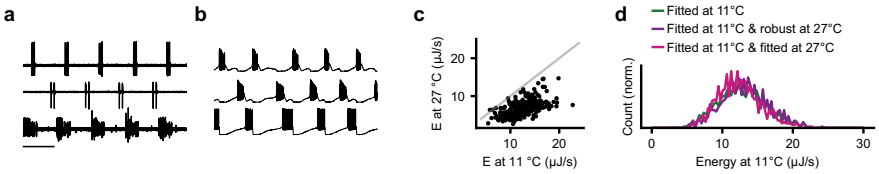
196

Fig. S4. Analysis of a second experimental preparation. (a) Experimental data recorded at 11°C. (b) Sample from posterior distribution matches the experimental data. (c) Energy consumption of 2804 model configurations that closely match experimental data. (d) Weights w of a linear regression from circuit parameters onto total energy consumption.



197

Fig. S5. Tuning neurons individually, for a second experimental preparation. (a) Black dots: Energy consumed by each neuron separately. Black line: Linear regression (correlation coefficient $r = -0.009$, p-value $p = 0.39$; LP versus PY, $r = 0.21$, $p = 0.0012$; AB/PD versus PY, $r = 0.059$, $p = 0.10$). (b) The activity produced by two parameter configurations produced with the strategy described in Fig. 6b. (c) A subset of the membrane (left) and synaptic (right) conductances for the configurations in panel (c). The membrane conductances are scaled with the following factors (left to right): 100, 10, 10,000, 100, 10,000. (d) Histogram over the total energy consumption of all 2804 model configurations in our database and the energy consumption of the configurations produced with the strategy described in Fig. 6b (red). (e) Histogram of the posterior log-probability for samples from the prior distribution (grey), for the 2804 models in our database (black), and for the configurations produced with the strategy described in Fig. 6b (red).



198

Fig. S6. Analysis of temperature robustness of a second experimental preparation. (a) Experimental data recorded at 27°C. (b) Sample from posterior distribution matches the experimental data. (c) Energy consumption at 11°C versus energy consumption at 27°C. (d) Green: Distribution of the energy consumption of circuits matching experimental data at 11°C. Purple: Distribution of the energy consumption of circuits that match data at 11°C and are robust at 27°C. Pink: Distribution of the energy consumption of circuits that match experimental data at 11°C and 27°C.

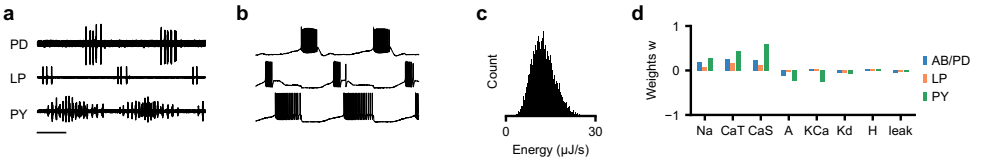


Fig. S7. **Analysis of a third experimental preparation.** (a) Experimental data recorded at 11°C. (b) Sample from posterior distribution matches the experimental data. (c) Energy consumption of 6926 model configurations that closely match experimental data. (d) Weights w of a linear regression from circuit parameters onto total energy consumption.

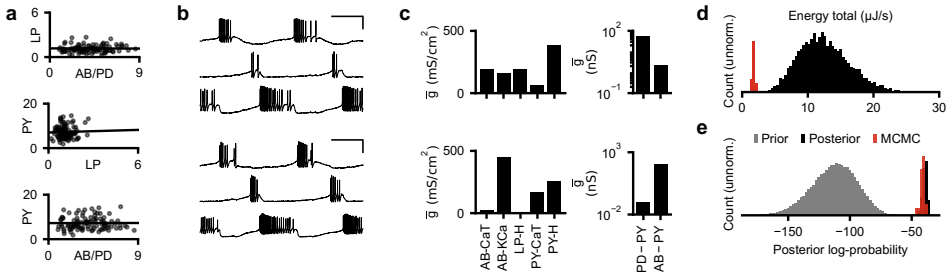
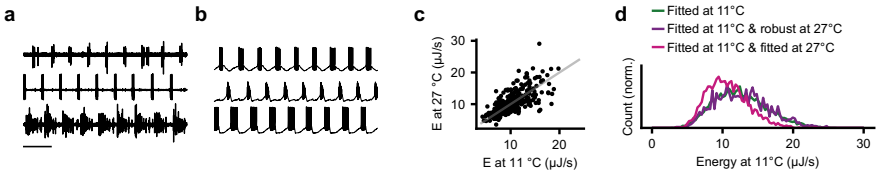


Fig. S8. Tuning neurons individually, for a third experimental preparation. (a) Black dots: Energy consumed by each neuron separately. Black line: Linear regression (correlation coefficient $r = -0.003$, p-value $p = 0.39$; LP versus PY, $r = 0.19$, $p = 0.007$; AB/PD versus PY, $r = 0.006$, $p = 0.77$). (b) The activity produced by two parameter configurations produced with the strategy described in Fig. 6b. (c) A subset of the membrane (left) and synaptic (right) conductances for the configurations in panel (c). The membrane conductances are scaled with the following factors (left to right): 100, 100, 10,000, 100, 10,000. (d) Histogram over the total energy consumption of all 6926 model configurations in our database and the energy consumption of the configurations produced with the strategy described in Fig. 6b (red). (e) Histogram of the posterior log-probability for samples from the prior distribution (grey), for the 6926 models in our database (black), and for the configurations produced with the strategy described in Fig. 6b (red).



201

Fig. S9. Analysis of temperature robustness of a third experimental preparation. (a) Experimental data recorded at 27°C. (b) Sample from posterior distribution matches the experimental data. (c) Energy consumption at 11°C versus energy consumption at 27°C. (d) Green: Distribution of the energy consumption of circuits matching experimental data at 11°C. Purple: Distribution of the energy consumption of circuits that match data at 11°C and are robust at 27°C. Pink: Distribution of the energy consumption of circuits that match experimental data at 11°C and 27°C.

202

Fig. S10. Experimental recording over an extended duration. Scale bar corresponds to 500 ms.

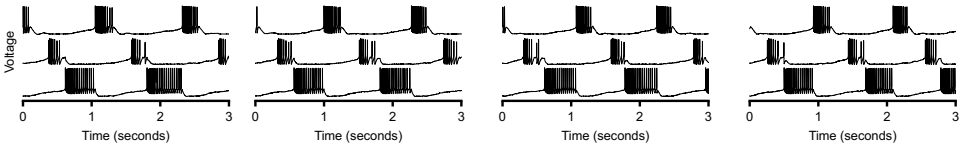


Fig. S11. Low variability of model simulations with the same underlying parameter set.

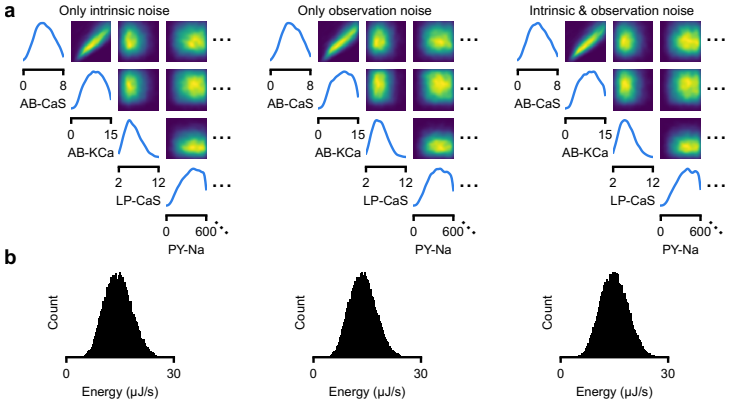
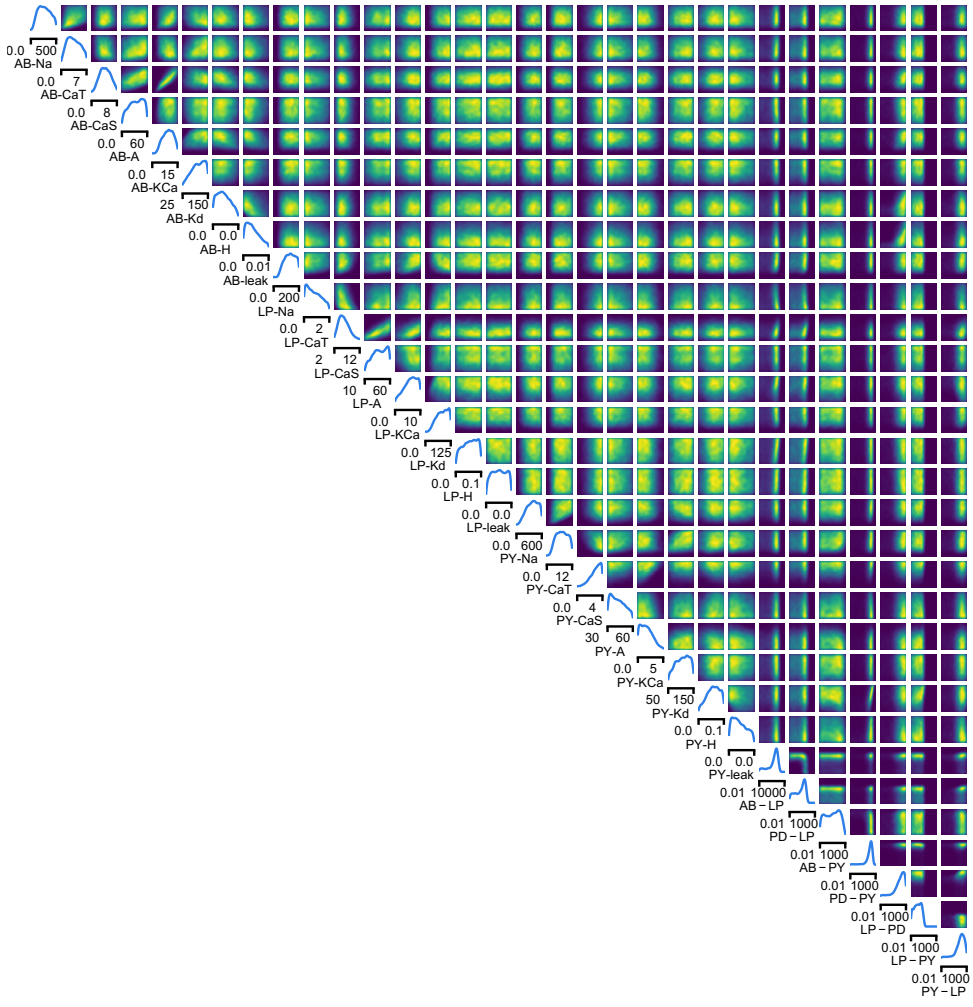
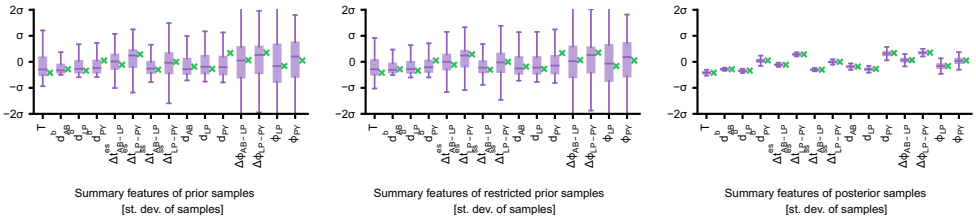


Fig. S12. Posterior distribution and range of energy consumptions are largely unaffected by the noise source. As intrinsic noise, we added Gaussian noise with mean zero and standard deviation 0.001 mV·ms$^{-0.5}$ at each time step. As observation noise, we added Gaussian noise with mean zero and standard deviation [10ms, 5ms, 5ms, 5ms, 0.005, 0.005, 0.005, 0.005, 0.005, 5ms, 5ms, 5ms, 5ms, 0.005, 0.005, 0.1ms, 0.1ms, 0.1ms] to the summary statistics. (a) Posterior distribution for each combination of noise sources. (b) Histogram of energy consumption of those model configurations that are sufficiently close to experimental data (see Methods for the closeness criterion).



205

Fig. S13. Full posterior distribution over circuit parameters given experimental data at 11°C. Panels on the diagonal are marginals, panels on the upper right are pairwise marginals. The first 24 parameters are membrane conductances, the last 7 parameters are synaptic conductances. All membrane conductances are maximal conductances and are given in mS/cm^2, all synaptic conductances are given in nS.



206

Fig. S14. Summary features of activity produced by sampling from the prior, the restricted prior, and the posterior. Experimentally observed activity in green. The boxplots indicate maximum, 75% quantile, median, 25% quantile, and minimum. All summary features are z-scored with the mean and standard deviation of all simulations from prior samples.

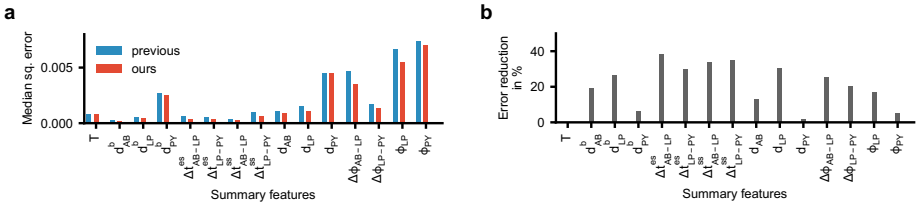


Fig. S15. Accuracy of the enhanced version of SNPE versus accuracy in (6). While we used half as many simulations (9 million versus 18 million), the accuracy of the method improved. (a) Median squared discrepancy between the experimentally measured activity and the activity produced by samples from the posterior. When using the classifier (red), the activity produced by posterior samples is closer to experimental activity than without the classifier (blue). (b) Reduction of mean squared discrepancy between our previous results and the presented method. All distances are computed after z-scoring the summary features with the mean and standard deviation of all prior samples.

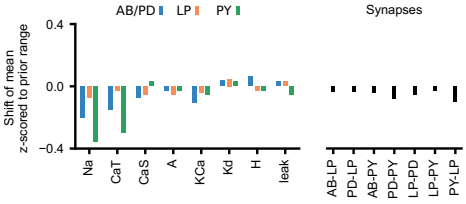


Fig. S16. Shift in the mean parameter value when enforcing low energy consumption. Difference between mean parameter value across all configurations in our database of 35,939 samples and mean parameter value across the 2% most efficient ones (divided by the parameter range of the prior).

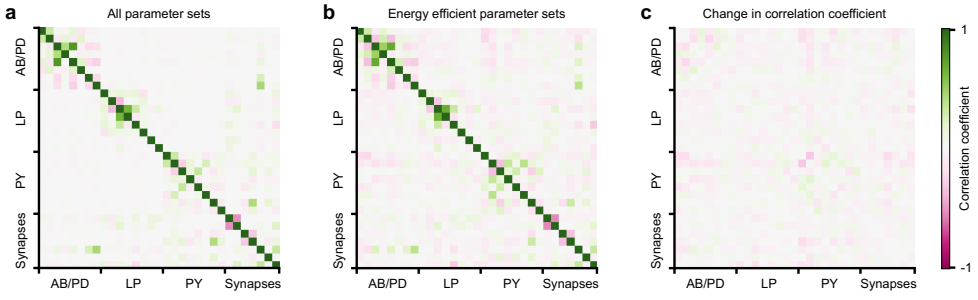


Fig. S17. **Change of parameter correlations when enforcing low energy consumption.** (a) Parameter correlation matrix of all 35,939 models in our database. (b) Parameter correlation matrix of the 2% most efficient models in our database. (c) Difference between correlation matrices shown in (a) and (b).

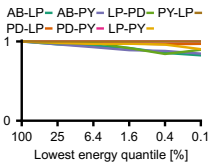
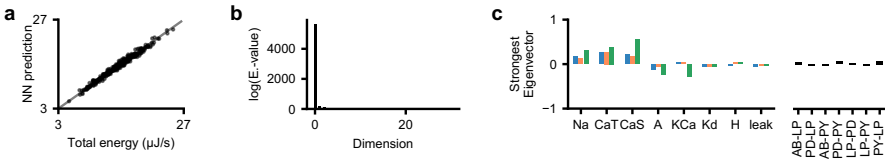
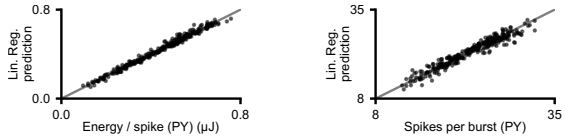


Fig. S18. Constraining synaptic conductances when enforcing low energy consumption. Standard deviation of synaptic parameters for models with energy consumption in the lowest 2% quantile divided by standard deviation of these parameters across all $35,939$ models in our database, evaluated for a range of quantiles.



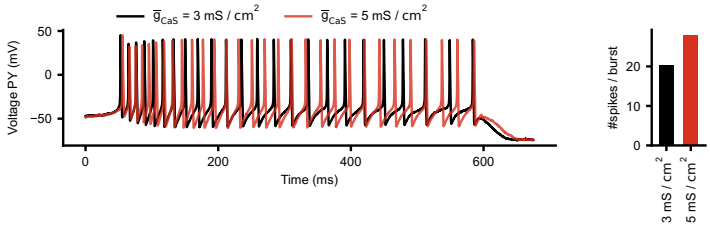
211

Fig. S19. **Neural network regression from circuit parameters onto the total energy consumption.** (a) Performance of a neural network predicting the total energy from circuit parameters. (b) Eigenvalue-spectrum of the trained neural network reveals a single dominating direction (details in Methods). (c) The eigenvector corresponding to the strongest eigenvalue is similar to the linear regression weights w (Fig. 4c).



212

Fig. S20. Performance of linear regression. Left: Performance of linear regression from circuit parameters (taken from our database of $35,393$ models) onto energy per spike in the PY neuron. Right: Performance of linear regression from circuit parameters onto the average number of spikes within a burst.



213

Fig. S21. Influence of maximal conductance of slow calcium current on circuit function. Left: Voltage trace in the PY neuron during activity produced by two circuit configurations (black and red) which are identical apart from the magnitude of $\overline{g}_{\text{CaS}}$. Right: The average number of spikes per burst in the PY neuron for the two configurations. The configuration with higher $\overline{g}_{\text{CaS}}$ produces more spikes.

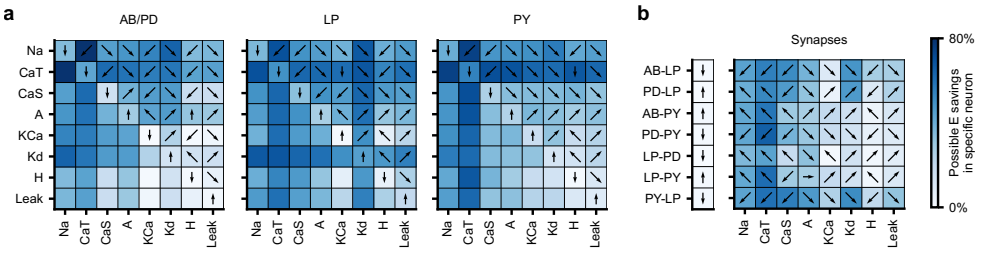
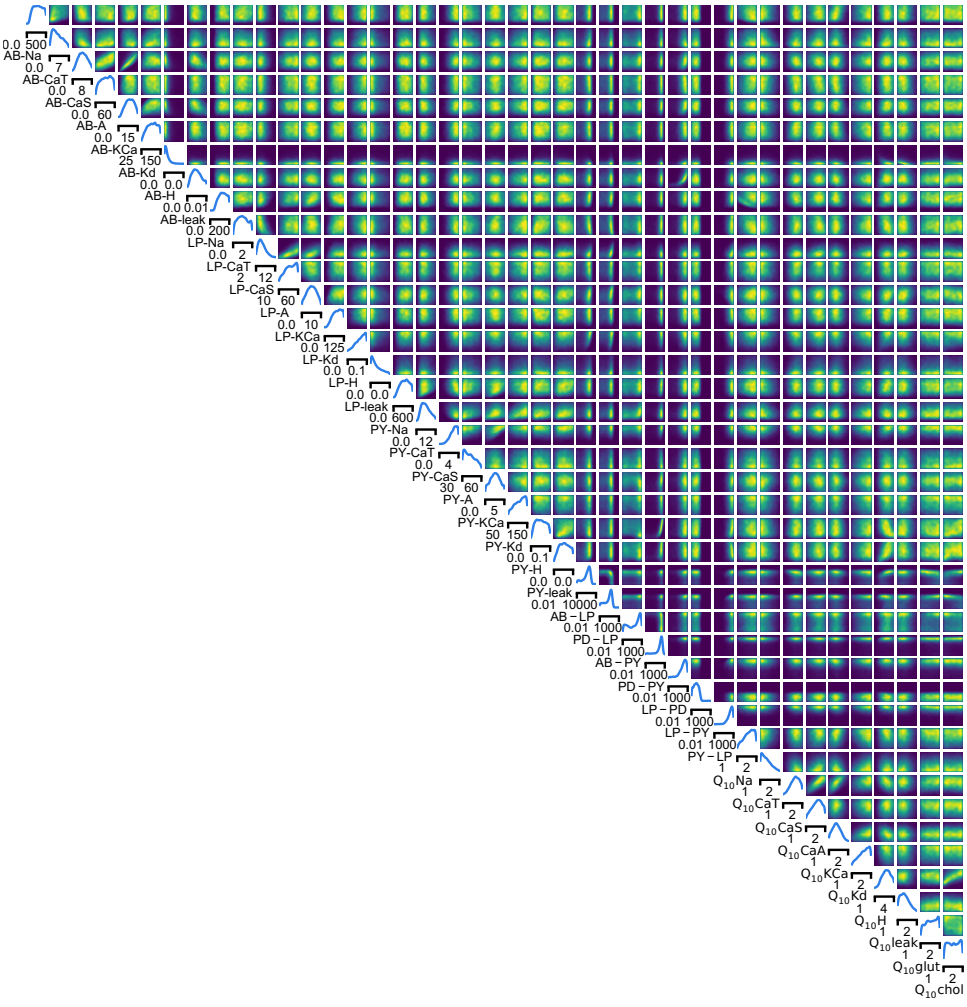


Fig. S22. Potential energy savings in a single neuron. (a) Fraction of energy that can be saved in a single neuron by modifying a single membrane parameter (diagonal of each matrix) or pairs of membrane parameters (upper and lower diagonal) in that neuron. Unlike Fig. 5e, only the energy in the neuron whose parameters are varied is considered. Colorbar as in panel (b). The arrows indicate the direction in which (pairs of) parameters should change in order to reduce energy: Left/right refers to the parameter on the x-axis, top/bottom refers to the parameter on the y-axis. (b) Fraction of energy that can be saved by modifying a synaptic conductance (vector on the left) or the synaptic conductance and one membrane conductance of the postsynaptic neuron (matrix on the right). The energy is computed in the postsynaptic neuron.



215

Fig. S23. Full posterior distribution over 31 circuit parameters and 10 Q_{10} parameters given experimental data at 11°C and 27°C.

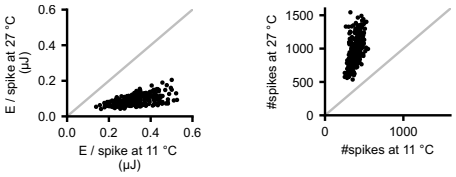


Fig. S24. Energy consumption of the circuit at 11°C and at 27°C. Energy per spike (left) and number of spikes (right) for parameter configurations simulated at 11°C and 27°C. The energy per spike is smaller at higher temperatures, but the number of spikes is higher at higher temperatures.

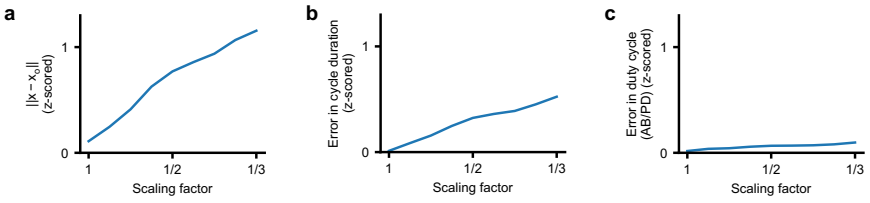


Fig. S25. **Scaling conductances with a constant factor does not produce activity that matches experimental data.** We selected the 20 most expensive circuit configurations from our database of 35,939 models and scaled all conductances by the same factor. The factor ranged from one to 1/3 (x-axis). All errors are z-scored by the standard deviation of prior predictive simulations. (a) As the conductances are scaled down, the average error of the model simulations to the observation increases. (b) The average error between the experimentally observed cycle duration and the simulations increases quickly as the conductances are scaled. (c) The average error between the experimentally observed duty cycle of the AB/PD neuron and the simulations increases minimally as the conductances are scaled.

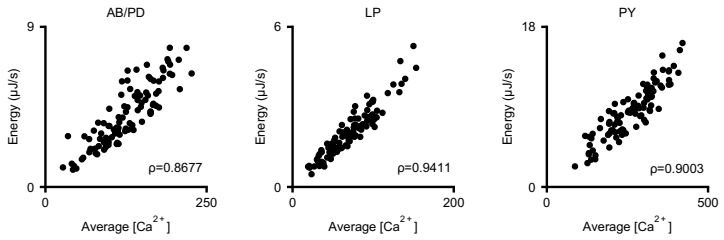


Fig. S26. Intracellular calcium level is linear related to energy consumption. For 100 circuit configurations within our database of 35,939 parameter configurations, we computed the time-average of the intracellular calcium concentration. This value correlates closely with time-averaged energy consumption. ρ is the Pearson correlation coefficient.

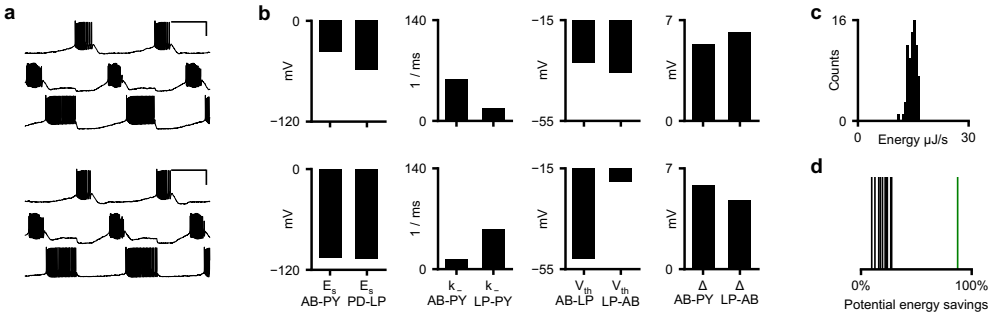


Fig. S27. Varying synaptic parameters affects energy consumption less strongly than varying maximal conductances. We kept all maximal conductances (including synaptic conductances) constant and varied all other 28 synaptic parameters (four values for each of the seven synapses: reversal potential of the synapse E_s, rate constant for transmitter-receptor dissociation rate k_-, half-activation voltage of the synapse V_{th}, and Δ determining the slope of the activation curve). (a) Two activities generated from disparate sets of synaptic parameters (but same maximal conductances). (b) Disparate synaptic parameters of the traces shown in (a). (c) Diversity of energy consumption of 41 configurations with the same maximal conductances but different synaptic conductances. These 41 configurations are the closest to the experimental data out of 50k sampled configurations. (d) We repeated the entire procedure for ten configurations of maximal conductances, taken from our database of $35,939$ configuration. The figure shows, for each such configuration of maximal conductances, the potential energy savings $(E_{\text{max}} - E_{\text{min}})/E_{\text{max}}$ obtained by varying the synaptic parameters. Green line is the potential energy savings when varying only maximal conductances (extracted from Fig. 2h).

References

1. SA Haddad, E Marder, Recordings from the c. borealis stomatogastric nervous system at different temperatures in the decentralized condition (2021).
2. SA Haddad, E Marder, Circuit robustness to temperature perturbation is altered by neuromodulators. *Neuron* **100**, 609–623 (2018).
3. AA Prinz, D Bucher, E Marder, Similar network activity from disparate circuit parameters. *Nat. Neurosci.* **7**, 1345 (2004).
4. L Abbott, E Marder, Modeling small networks (1998).
5. JH Goldwyn, E Shea-Brown, The what and where of adding channel noise to the hodgkin-huxley equations. *PLoS computational biology* **7**, e1002247 (2011).
6. PJ Gonçalves, et al., Training deep neural density estimators to identify mechanistic models of neural dynamics. *Elife* **9**, e56261 (2020).
7. LM Alonso, E Marder, Temperature compensation in a small rhythmic circuit. *Elife* **9**, e55470 (2020).
8. B Frankenhaeuser, L Moore, The effect of temperature on the sodium and potassium permeability changes in myelinated nerve fibres of xenopus laevis. *The J. Physiol.* **169**, 431 (1963).
9. LS Tang, et al., Precise temperature compensation of phase in a rhythmic motor pattern. *PLoS Biol* **8**, e1000469 (2010).
10. JS Caplan, AH Williams, E Marder, Many parameter sets in a multicompartment model oscillator are robust to temperature perturbations. *J. Neurosci.* **34**, 4963–4975 (2014).
11. A Moujahid, A d'Anjou, F Torrealdea, F Torrealdea, Energy and information in hodgkin-huxley neurons. *Phys. Rev. E* **83**, 031912 (2011).
12. B Sengupta, M Stemmler, SB Laughlin, JE Niven, Action potential energy efficiency varies among neuron types in vertebrates and invertebrates. *PLoS Comput. Biol* **6**, e1000840 (2010).
13. A Hasenstaub, S Otte, E Callaway, TJ Sejnowski, Metabolic cost as a unifying principle governing neuronal biophysics. *Proc. Natl. Acad. Sci.* **107**, 12329–12334 (2010).
14. D Bray, *Cell movements: from molecules to motility*. (Garland Science), (2000).
15. C Durkan, A Bekasov, I Murray, G Papamakarios, Neural spline flows in *Advances in Neural Information Processing Systems*. pp. 7511–7522 (2019).
16. F Pedregosa, et al., Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011).
17. K He, X Zhang, S Ren, J Sun, Deep residual learning for image recognition in *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 770–778 (2016).
18. N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting. *The journal machine learning research* **15**, 1929–1958 (2014).
19. PG Constantine, *Active subspaces: Emerging ideas for dimension reduction in parameter studies*. (SIAM), (2015).
20. RM Neal, Slice sampling. *The annals statistics* **31**, 705–767 (2003).

Truncated proposals for scalable and hassle-free simulation-based inference

Michael Deistler
University of Tübingen
michael.deistler@uni-tuebingen.de

Pedro J Gonçalves*
University of Tübingen
pedro.goncalves@uni-tuebingen.de

Jakob H Macke*
University of Tübingen
Max Planck Institute for Intelligent Systems
jakob.macke@uni-tuebingen.de

Abstract

Simulation-based inference (SBI) solves statistical inverse problems by repeatedly running a stochastic simulator and inferring posterior distributions from model-simulations. To improve simulation efficiency, several inference methods take a sequential approach and iteratively adapt the proposal distributions from which model simulations are generated. However, many of these sequential methods are difficult to use in practice, both because the resulting optimisation problems can be challenging and efficient diagnostic tools are lacking. To overcome these issues, we present Truncated Sequential Neural Posterior Estimation (TSNPE). TSNPE performs sequential inference with truncated proposals, sidestepping the optimisation issues of alternative approaches. In addition, TSNPE allows to efficiently perform coverage tests that can scale to complex models with many parameters. We demonstrate that TSNPE performs on par with previous methods on established benchmark tasks. We then apply TSNPE to two challenging problems from neuroscience and show that TSNPE can successfully obtain the posterior distributions, whereas previous methods fail. Overall, our results demonstrate that TSNPE is an efficient, accurate, and robust inference method that can scale to challenging scientific models.

1 Introduction

Computational models are an important tool to understand physical processes underlying empirically observed phenomena. These models, often implemented as numerical *simulators*, incorporate mechanistic knowledge about the physical process underlying data generation, and thereby provide an interpretable model of empirical observations. In many cases, several parameters of the simulator have to be inferred from data, e.g., with Bayesian inference. However, performing Bayesian inference in these models can be difficult: Running the simulator may be computationally expensive, evaluating the likelihood-function might be computationally infeasible, and the model might not be differentiable. In order to overcome these limitations, Approximate Bayesian Computation (ABC) methods [Beaumont et al., 2002, 2009], synthetic likelihood approaches [Wood, 2010], and neural network-based methods [e.g., Papamakarios and Murray, 2016, Hermans et al., 2020, Thomas et al., 2022] have been developed.

*Equal contribution

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

A subset of neural network-based methods, known as neural posterior estimation (NPE) [Papamakarios and Murray, 2016, Lueckmann et al., 2017, Greenberg et al., 2019], train a neural density estimator on simulated data such that the density estimator directly approximates the posterior. Unlike other methods, NPE does not require any further Markov-chain Monte-Carlo (MCMC) or variational inference (VI) steps. As it provides an *amortized* approximation of the posterior, which can be used to quickly evaluate and sample the approximate posterior for any observation, NPE allows the application in time-critical and high-throughput inference scenarios [Gonçalves et al., 2020, Radev et al., 2020, Dax et al., 2021], and fast application of diagnostic methods which require posterior samples for many different observations [Cook et al., 2006, Talts et al., 2018]. In addition, unlike methods targeting the likelihood (e.g., neural likelihood estimation, NLE [Papamakarios et al., 2019, Lueckmann et al., 2019]), NPE can learn summary statistics from data and it can use equivariances in the simulations to improve the quality of inference [Dax et al., 2021, 2022].

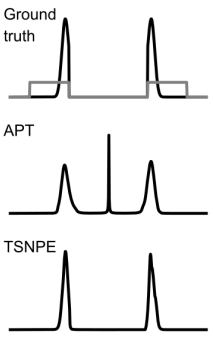


Figure 1: **APT vs TSNPE.** Top: Prior (gray) and true posterior (black). APT matches true posterior within the prior bounds but 'leaks' into region without prior support. TSNPE (ours) matches true posterior.

If inference is performed for a particular observation $\mathbf{x}_o$, sampling efficiency of NPE can be improved with *sequential* training schemes: Instead of drawing parameters from the prior distribution, they are drawn adaptively from a proposal (e.g., a posterior estimate obtained with NPE) in order to optimize the posterior accuracy for a particular $\mathbf{x}_o$. These procedures are called Sequential Neural Posterior Estimation (SNPE) [Papamakarios and Murray, 2016, Lueckmann et al., 2017, Greenberg et al., 2019] and have been reported to be more simulation-efficient than training the neural network only on parameters sampled from the prior, across a set of benchmark tasks [Lueckmann et al., 2021].

Despite the potential to improve simulation-efficiency, two limitations have impeded a more widespread adoption of SNPE by practitioners: First, the sequential scheme of SNPE can be unstable. SNPE requires a modification of the loss function compared to NPE, which suffers from issues that can limit its effectiveness on (or even prevent their application to) complex problems (see Sec. 2). Second, several commonly used diagnostic tools for SBI [Talts et al., 2018, Miller et al., 2021, Hermans et al., 2021] rely on performing inference across multiple observations. In SNPE (in contrast to NPE), this requires generating new simulations and network retraining for each observation, which often prohibits the use of such diagnostic tools [Lueckmann et al., 2021, Hermans et al., 2021].

Here, we introduce Truncated Sequential Neural Posterior Estimation (TSNPE) to overcome these limitations. TSNPE follows the SNPE formalism, but uses a proposal which is a *truncated* version of the prior: TSNPE draws simulations from the prior, but rejects them *before simulation* if they lie outside of the support of the approximate posterior. Thus, the proposal is (within its support) proportional to the prior, which allows us to train the neural network with maximum-likelihood in every round and, therefore, sidesteps the instabilities (and hence 'hassle') of previous SNPE methods. Our use of truncated proposals is strongly inspired by Blum and François [2010] and Miller et al. [2020, 2021], who proposed truncated proposals respectively for regression-adjustment approaches in ABC and for neural ratio estimation (see Discussion). Unlike methods based on likelihood(-ratio)-estimation [Miller et al., 2021, Hermans et al., 2021], TSNPE allows direct sampling and density evaluation of the approximate posterior, and thus permits computing expected coverage of the full posterior quickly (without MCMC) and at every iteration of the algorithm, thus allowing to diagnose failures of the method even for high-dimensional parameter spaces (we term this 'simulation-based coverage calibration' (SBCC), given its close connection with simulation-based calibration, SBC, Cook et al. [2006], Talts et al. [2018]).

We show that TSNPE is as efficient as the SNPE method 'Automatic Posterior Transformation' (APT, Greenberg et al. [2019]) on several established benchmark problems (Sec. 4.1). We then demonstrate that for two challenging neuroscience problems, TSNPE—but not APT—can robustly identify the posterior distributions (Sec. 4.2).

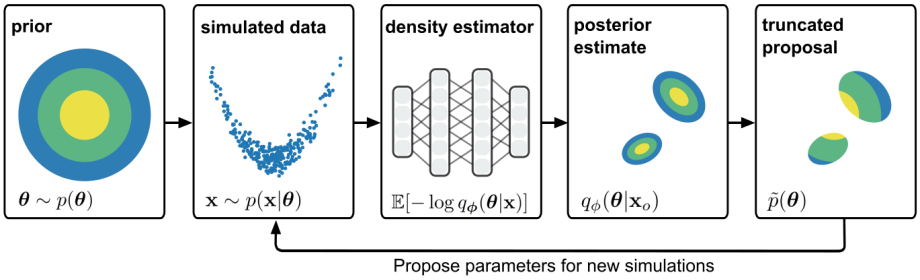


Figure 2: **Truncated Sequential Neural Posterior Estimation (TSNPE).** The method starts by sampling from the prior, running the simulator, and training a neural density estimator with maximum-likelihood to approximate the posterior. In subsequent rounds, parameters are sampled from the prior, but rejected if they lie outside of the support of the approximate posterior. With these proposals, the neural density estimator can be trained with maximum-likelihood in all rounds.

2 Background

In Neural Posterior Estimation (NPE), parameters are sampled from the prior $p(\boldsymbol{\theta})$ and simulated (i.e., $\mathbf{x}$ is sampled from $p(\mathbf{x}|\boldsymbol{\theta})$). Then, a neural density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ (in our case a normalizing flow), with learnable parameters $\boldsymbol{\phi}$, is trained to minimize the loss:

$$\min_{\boldsymbol{\phi}} \mathcal{L} = \min_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta})} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})} [-\log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})],$$

which is minimized if and only if, for a sufficiently expressive density estimator, $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}) = p(\boldsymbol{\theta}|\mathbf{x})$ for all $\mathbf{x} \in \text{supp}(p(\mathbf{x}))$ [Paige and Wood, 2016, Papamakarios and Murray, 2016]. Throughout this study, we refer to training with this loss function as maximum-likelihood training, although the neural density estimator targets the posterior directly.

Sequential Neural Posterior Estimation (SNPE) aims to infer the posterior distribution $p(\boldsymbol{\theta}|\mathbf{x}_o)$ for a particular observation $\mathbf{x}_o$. SNPE initially performs NPE and, thereby, obtains an initial estimate of the posterior distribution. It then samples parameters from a proposal $\tilde{p}(\boldsymbol{\theta})$, which is often chosen to be the previously obtained estimate of the posterior $\tilde{p}(\boldsymbol{\theta}) = q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)$, and retrains the neural density estimator [Papamakarios and Murray, 2016]. This procedure can be repeated for several rounds.

Importantly, if parameters $\boldsymbol{\theta}$ are sampled from the proposal $\tilde{p}(\boldsymbol{\theta})$ rather than from the prior $p(\boldsymbol{\theta})$, the estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ that minimizes the maximum-likelihood loss function no longer converges to the true posterior. If one used the maximum-likelihood loss on data sampled from $\tilde{p}(\boldsymbol{\theta})$, i.e., $\mathcal{L} = \mathbb{E}_{\boldsymbol{\theta} \sim \tilde{p}(\boldsymbol{\theta})} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})} [-\log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})]$, then $\mathcal{L}$ would be minimized by $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}) \propto p(\boldsymbol{\theta}|\mathbf{x}) \frac{\tilde{p}(\boldsymbol{\theta})}{p(\boldsymbol{\theta})}$, which is not the true posterior. Multiple schemes have been developed to overcome this [Papamakarios and Murray, 2016, Lueckmann et al., 2017]. The most recent of these methods, Automatic Posterior Transformation (APT, or SNPE-C, in its atomic version) [Greenberg et al., 2019, Durkan et al., 2020] employs a loss that aims to classify the parameter set that generated a particular data point among other parameter sets (details in Appendix Sec. 6.5).

While APT has been reported to significantly outperform previous methods, several studies have also described cases in which the approach exhibits performance issues: Both the original APT paper [Greenberg et al., 2019] and Durkan et al. [2020] reported that APT can show 'leakage' of posterior mass outside of bounded priors. We demonstrate this issue on a simple 1-dimensional simulator with bounded prior (Fig. 1, Appendix Fig. 7). The posterior estimated by APT is only required to match the true posterior density *within the support of the prior* (details in Appendix Sec. 6.5). Thus, after five rounds of APT, while the approximate posterior matches the true posterior within the bounds of the prior, a substantial fraction of posterior mass lies in regions with zero prior probability. In simple models, approximate posterior samples that lie outside of the prior bounds can be efficiently rejected. However, in models with high numbers of parameters, the rejection rate can become so large that drawing posterior samples which lie inside of the prior bounds is prohibitive. For example, Glöckler et al. [2022] reported a rejection rate of more than 99.9999% in a model with 31 parameters, thus requiring approximately one minute to draw a single posterior sample from within the prior bounds.

Algorithm 1: TSNPE

Inputs: prior $p(\boldsymbol{\theta})$, observation $\mathbf{x}_o$, simulations per round N, number of rounds R, ϵ that defines the highest-probability region (HPR$_\epsilon$)
Outputs: Approximate posterior $q_{\boldsymbol{\phi}}$.
Initialize: Proposal $\tilde{p}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$, dataset $\mathcal{X} = \{\}$
for $r \in [1, ..., R]$ **do**
  for $i \in [1, ..., N]$ **do**
    $\boldsymbol{\theta}_i \sim \tilde{p}(\boldsymbol{\theta})$
    simulate $\mathbf{x}_i \sim p(\mathbf{x}|\boldsymbol{\theta}_i)$
    add $(\boldsymbol{\theta}_i, \mathbf{x}_i)$ to $\mathcal{X}$
  $\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} -\frac{1}{N} \sum_{(\boldsymbol{\theta}_i, \mathbf{x}_i) \in \mathcal{X}} \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_i|\mathbf{x}_i)$
  Compute expected coverage($\tilde{p}(\boldsymbol{\theta}), q_{\boldsymbol{\phi}}$) ; `// see Alg. 2`
  $\tilde{p}(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}) \cdot \mathbb{1}_{\boldsymbol{\theta} \in \text{HPR}_\epsilon}$; `// see Alg. 3`

We overcome these limitations by using 'truncated' proposal distributions. This allows us to train with maximum-likelihood at every round, thereby sidestepping issues of previous SNPE methods.

3 Methodology

3.1 Truncated proposals for SNPE

Given a particular observation $\mathbf{x}_o$, we suggest to restrict the proposals $\tilde{p}(\boldsymbol{\theta})$ to be proportional to the prior $p(\boldsymbol{\theta})$ at least in the $1 - \epsilon$ highest-probability-region (HPR$_\epsilon$, the smallest region that contains $1 - \epsilon$ of the mass) of $p(\boldsymbol{\theta}|\mathbf{x}_o)$, i.e.

$$\tilde{p}(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}) \cdot \mathbb{1}_{\boldsymbol{\theta} \in \mathcal{M}}$$

with HPR$_\epsilon(p(\boldsymbol{\theta}|\mathbf{x}_o)) \subseteq \mathcal{M}$. Thus, $\tilde{p}(\boldsymbol{\theta})$ is a 'truncated' proposal. The key insight is that, when using such a proposal and $\epsilon = 0$, one can train $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ with maximum likelihood:

$$\min_{\boldsymbol{\phi}} \mathcal{L} = \min_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\theta} \sim \tilde{p}(\boldsymbol{\theta})} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})}[-\log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})],$$

and $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)$ will still converge to $p(\boldsymbol{\theta}|\mathbf{x}_o)$ (Proof in Appendix Sec. 6.2).

We estimate $\mathcal{M}$ as the HPR$_\epsilon$ of the approximate posterior $\mathcal{M} = \text{HPR}_\epsilon(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o))$. Since the maximum-likelihood loss employed to train $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ is support-covering, the HPR$_\epsilon$ of $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)$ tends to cover the HPR$_\epsilon$ of $p(\boldsymbol{\theta}|\mathbf{x}_o)$ [Bishop and Nasrabadi, 2006].

In order to obtain the HPR$_\epsilon$ of $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)$, we define a threshold $\boldsymbol{\tau}$ on the approximate posterior density $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)$. To do so, we use a normalizing flow as $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$, which allows for closed-form density evaluation and fast sampling. We then approximate the HPR$_\epsilon$ of $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)$ as

$$\text{HPR}_\epsilon(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)) \approx \mathbb{1}_{q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o) > \boldsymbol{\tau}}.$$

We chose $\boldsymbol{\tau}$ as the ϵ-quantile of approximate posterior densities of samples from $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)$, and evaluated TSNPE for $\epsilon = 10^{-3}$, 10^{-4}, and 10^{-5}. Values of $\epsilon > 0$ yield a proposal prior which has smaller support than the current estimate of the posterior, e.g., using $\epsilon = 10^{-3}$ neglects 0.1% of mass from the approximate-posterior support. Thus, this approach leads to errors in posterior estimation, e.g., to 'under-covered' posteriors (Appendix Sec. 6.10). However, empirically, the error induced by this truncation is negligible, as we will demonstrate on several benchmark tasks. We note that TSNPE can be trained on data pooled from all rounds (Appendix Sec. 6.2). TSNPE is summarized in Alg. 1 (Fig. 2).

3.2 Sampling from the truncated proposal

To generate training data for subsequent rounds, we have to draw samples from the truncated proposal $\tilde{p}(\boldsymbol{\theta})$, and here we explored rejection sampling and sampling importance resampling (SIR) [Rubin, 1988]. For rejection sampling, we sample the prior $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$ and accept samples only if their probability under the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ is above threshold $\boldsymbol{\tau}$.

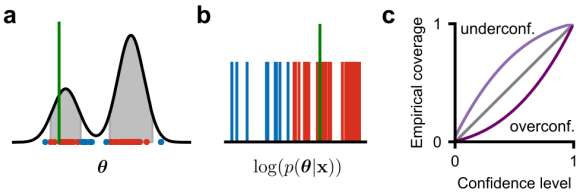


Figure 3: **Diagnostic tool.** (a) Parameter θ^* (green) lies within the 1-α confidence region (gray) of the estimated posterior. (b) $\log(p(\theta^*|x))$ is above the 1-α quantile of posterior samples. (c) 1-α versus empirical coverage, averaged over θ^*.

This strategy samples from the truncated proposal exactly, but can fail if the rejection rate becomes too high. To deal with these situations, we used SIR. For each sample from the truncated proposal, SIR draws K samples from the approximate posterior, computes weights $w_{i=1...K} = p(\boldsymbol{\theta}_i)\mathbb{1}_{\boldsymbol{\theta}_i \in \mathcal{M}}/q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_i|\mathbf{x})$, normalizes w_i such that they sum to one, draws from a categorical distribution with weights $s \sim \text{Categorical}(w_i)$, and selects the posterior sample with index s. SIR requires a fixed sampling budget of K posterior samples per sample from the truncated proposal and returns exact samples from the truncated proposal for $K \to \infty$. Too low values of K lead to too narrow proposals and posterior approximations. When run for a number of rounds, this behaviour reinforces itself and can lead to divergence of TSNPE (Appendix Fig. 13). We, thus, chose a high value $K = 1024$. In our experiments, we did not observe poor SIR performance, but we emphasise the importance of using tools to diagnose potential failures of TSNPE (see below) or SIR (e.g. by inspecting the effective sample size, Appendix Sec. 6.12). When SIR fails, methods such as nested sampling, adaptive multi-level splitting, or sequential Monte-Carlo sampling could be viable alternatives [Skilling, 2004, Cérou and Guyader, 2007, Doucet et al., 2001]. We discuss computational costs of rejection sampling and SIR in Appendix Sec. 6.11.

3.3 Coverage diagnostic

In order for the estimated posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)$ to converge to $p(\boldsymbol{\theta}|\mathbf{x}_o)$, TSNPE requires $\text{supp}(p(\boldsymbol{\theta}|\mathbf{x}_o)) \subseteq \text{HPR}_\epsilon(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o))$, i.e., the estimated posterior must be broader than the true posterior (proof in Appendix Sec. 6.2). In order to diagnose whether the posterior is, on average, sufficiently broad, we perform expected coverage tests as proposed in Dalmasso et al. [2020], Miller et al. [2021], Hermans et al. [2021].

As described in Dalmasso et al. [2020], Rozet et al. [2021] and illustrated in Fig. 3, the coverage of the approximate posterior can be computed as

$$1 - \alpha = \int q_\phi(\boldsymbol{\theta}|\mathbf{x}^*)\mathbb{1}(q_\phi(\boldsymbol{\theta}^*|\mathbf{x}^*) \geq q_\phi(\boldsymbol{\theta}|\mathbf{x}^*))d\boldsymbol{\theta}$$

where $\boldsymbol{\theta}^*$ is sampled from the truncated proposal and $\mathbf{x}^*$ is the corresponding simulator output. In order to approximate this integral, one has to either evaluate the approximate posterior on a grid [Dalmasso et al., 2020, Hermans et al., 2021] or apply a Monte-Carlo average which includes repeatedly sampling (and evaluating) the (unnormalized) approximate posterior [Miller et al., 2021, Rozet et al., 2021]. The first option does not scale to high-dimensional spaces whereas the second is computationally expensive for methods estimating likelihood(-ratios) and, thus, require MCMC. In contrast, the TSNPE-posterior can be sampled from and evaluated in closed-form, leading to a computationally efficient and scalable diagnostic which can be run after every training round.

Expected coverage can be computed as an average of the coverage across multiple pairs $(\boldsymbol{\theta}^*, \mathbf{x}^*)$ [Miller et al., 2021, Hermans et al., 2021] and should match the confidence level for all confidence levels $(1 - \alpha) \in [0, 1]$ (Fig. 3c). We term this procedure of computing the empirical coverage 'simulation-based coverage calibration' (SBCC), due to its close connection with SBC [Cook et al., 2006, Talts et al., 2018] (identical under certain conditions, Appendix Sec. 6.6). For TSNPE, it is important that the empirical expected coverage matches the confidence level for high confidence levels (i.e., for small α), since overconfidence in these regions would indicate that ground-truth parameters $\boldsymbol{\theta}^*$ are falsely excluded from the $\text{HPR}_{\epsilon=\alpha}$. SBCC is summarized in Appendix Alg. 2.

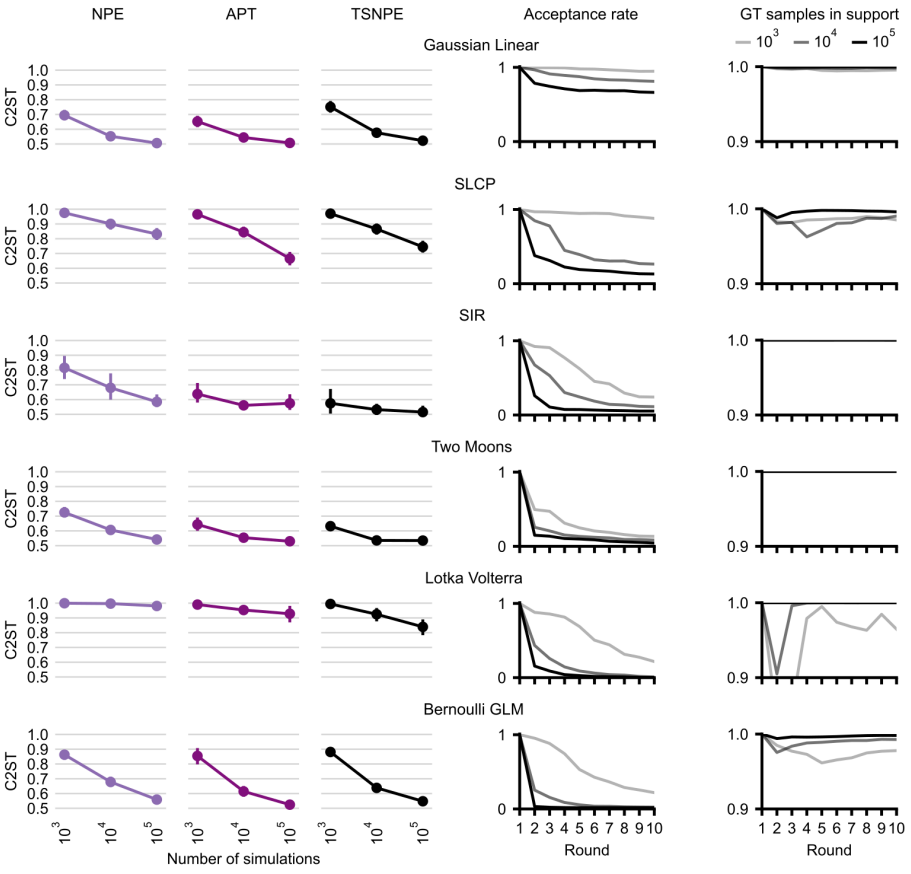


Figure 4: **Performance on six benchmark tasks.** Left three columns: classifier two-sample test accuracy (C2ST) of NPE (left), APT (middle), and TSNPE (right) for three simulation budgets. Forth column: Fraction of prior samples within the approximate-posterior HPR_ϵ in each round for each simulation budget. Fifth column: Fraction of true-posterior samples within the approximate-posterior HPR_ϵ. TSNPE with $\epsilon = 10^{-4}$ and rejection sampling from truncated proposal.

4 Results

We evaluated TSNPE on several benchmark tasks and on two complex problems from neuroscience. We found that TSNPE performs as well as APT on the benchmark tasks and that it is robust to choices of ϵ. In addition, we found that, in contrast with APT, TSNPE can successfully infer the posterior distribution for complex models with large numbers of parameters.

4.1 Performance on benchmark tasks

We compared TSNPE with NPE and APT on six benchmark tasks for which samples from the ground-truth posterior are available (see Appendix Sec. 6.9 for tasks) [Lueckmann et al., 2021]. We quantified the performance with a classifier two-sample test (C2ST), for which 0.5 indicates that the approximate posterior is identical to the ground-truth posterior, whereas 1.0 implies that the distributions can be completely separated by a classifier. Overall, APT and TSNPE perform similarly well and both outperform NPE (Fig. 4, left three columns). On two of the six tasks (Gaussian Linear and SLCP), APT has slightly better performance than TSNPE, whereas on two other tasks (SIR

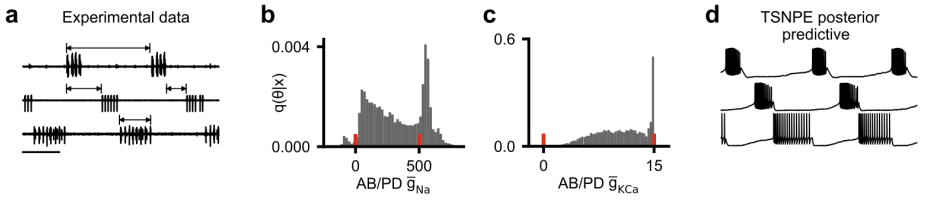


Figure 5: **Pyloric network inference.** (a) Data [Haddad and Marder, 2021]. (b) APT approximate posterior (1D-marginal) exhibits leakage (red: prior bounds). (c) APT approximate posterior when forcing the density estimator into constrained space. The spike at the upper prior bound is at odds with previously published posterior distributions [Gonçalves et al., 2020, Deistler et al., 2021, Glöckler et al., 2022] and produces poor predictive samples (Appendix Fig. 19). (d) TSNPE posterior predictive sample matches summary statistics of the experimental data.

and Lotka-Volterra), TSNPE outperforms APT. Overall, TSNPE and APT perform similarly well, demonstrating that TSNPE is competitive with previous methods on benchmark tasks.

In order to get insights into the improved performance of TSNPE as compared to NPE, we computed the fraction of prior samples that lie within the HPR_ϵ of the approximate posterior (Fig. 4, fourth column). In tasks with broad posteriors and few simulations, the HPR_ϵ is almost as wide as the prior and thus the performances of NPE and TSNPE are similar (e.g., SLCP with 1k simulations). In other tasks and with more simulations, the HPR_ϵ is much narrower than the prior, leading to an improvement in simulation efficiency (e.g., Lotka-Volterra with 100k simulations).

Finally, we evaluated whether the HPR_ϵ of the approximate posterior contains the support of the true posterior (Fig. 4, fifth column). We computed the fraction of true-posterior samples within the HPR_ϵ of the approximate posterior. For most tasks, fewer than 0.1% of samples were excluded, and the rate of erroneously rejected samples decreased as more simulations were used. In the Lotka-Volterra task with 1k and 10k simulations, many ground-truth samples were rejected and TSNPE performed poorly, but NPE and APT also failed to solve the task. Thus, while truncated proposals can potentially induce posterior biases, these only have a negligible effect on the performance of TSNPE. We note that TSNPE performance is qualitatively unaffected by the choice of $\epsilon \leq 10^{-4}$ and proposal sampling scheme (Appendix Fig. 8, Fig. 9, Fig.10). Applying truncated proposals to APT leads to equally good or worse performance than 'standard' APT, depending on the task (Appendix Fig. 14).

4.2 Efficient and robust inference in two complex neuroscience problems

Next, we evaluate the performance of TSNPE on two challenging neuroscience problems, where the competitive advantage of TSNPE is fully realized.

Pyloric network We applied TSNPE to a challenging real-world simulator from neuroscience: The pyloric network of the stomatogastric ganglion in the crab *Cancer Borealis* [Prinz et al., 2003, 2004]. The model has 31 parameters and simulates 3 voltage traces that we reduce to 18 summary statistics. The prior distribution is uniform within previously described parameter ranges [Prinz et al., 2004, Gonçalves et al., 2020]. We identify the posterior distribution given experimentally observed data [Haddad and Marder, 2021] (Fig. 5a) with APT and TSNPE (13 rounds, 30k simulations per round).

When applying APT 'out of the box' (from 'sbi' toolbox [Tejero-Cantero et al., 2020]), the rate of approximate-posterior samples within the prior bounds was 0.02% after the second round and 0.0000% after the third round (Fig. 5b), which rendered a fourth round too computationally expensive.

We attempted to overcome these issues by appending a transformation T to the density estimator $q_\phi(\boldsymbol{\theta}|\mathbf{x})$ such that its support is constrained to match the support of the prior. In practice, we used a sigmoid transformation. While the resulting approximate posterior exhibited no leakage, this setup revealed another problem when running APT: In transformed (i.e., unbounded) space, the density estimator $q_\phi(\boldsymbol{\theta}|\mathbf{x})$ can put significant mass in regions outside of the training data. When forced into constrained space, these 'leaking' regions lead to spikes at the bounds of the parameter space (Fig. 5c, further details in Appendix Sec. 6.5; illustration, additional tests and full posterior in Appendix

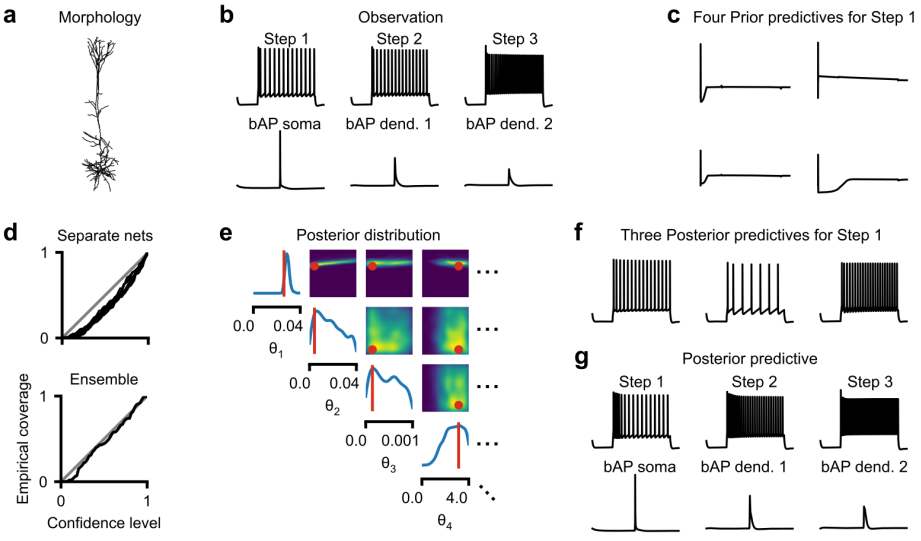


Figure 6: **TSNPE on L5PC.** (a) Cell morphology. (b) Observation. (c) Four prior samples for the Step 1 protocol. (d) Coverage for 1 and 10 neural nets. (e) Posterior. True parameter in red. (f) Three posterior predictives for the Step 1 protocol. (g) One posterior predictive for all protocols.

Figs. 17, 16 and 18). These spikes are at odds with previously published posterior distributions [Gonçalves et al., 2020, Deistler et al., 2021, Glöckler et al., 2022] and samples from these parameter regions do not produce good simulations (Appendix Fig. 19). This demonstrates that leakage occurs in APT even when the density estimator is forced into constrained space and that these issues lead to an incorrect posterior approximation as well as to poor predictive samples.

We applied TSNPE to this task for 13 rounds without any issue. The resulting posterior produces samples that closely match the observed data (Fig. 5d, more samples in Appendix Fig. 15, posterior distribution across all 31 parameters in Appendix Fig. 20). The obtained posterior is similar to previously published posteriors [Gonçalves et al., 2020, Deistler et al., 2021, Glöckler et al., 2022].

Multicompartment model of a single neuron Finally, we turn to a landmark problem in neuroscience for which the posterior has not yet been identified: A morphologically detailed model of a thick-tufted layer 5 pyramidal cell (L5PC) from the neocortex [Ramaswamy et al., 2015, Markram et al., 2015, Van Geit et al., 2016]. The model describes the response of a neuron to current stimuli of different strengths. The model has approximately 7000 separate compartments which compose the anatomy of the cell (Fig. 6a). Each compartment has dynamics based on the Hodgkin-Huxley equations [Hodgkin and Huxley, 1952] and contains multiple ion channels (details in Van Geit et al. [2016]). The model has 20 free parameters which are the maximal channel conductances and time constants of the ion channels. The simulation consists of four separate simulations corresponding to experimental protocols which describe the voltage response to different stimuli. For the first three protocols (Step 1, Step 2, Step 3), each voltage response is characterized by 10 summary statistics. The fourth protocol models the back-propagation of the voltage response through the dendritic tree and is captured by 5 additional summary statistics (bAP soma, bAP dend. 1, bAP dend. 2). In total, the model produces 35 summary statistics, to which we add Gaussian noise with diagonal covariance matrix capturing the response variability of previously reported measurements [Hay et al., 2011].

Our goal is to infer the posterior distribution over 20 parameters given 35 summary statistics that were simulated—and thus have a known ground-truth parameter set— to resemble experimentally observed activity (Fig. 6b). The prior is a uniform distribution within previously established bounds [Van Geit et al., 2016]. A major difficulty in fitting this model is that a large fraction of prior samples generate summary statistics that are very different from the observed data: In particular, about 99.98% of prior predictives contain at least one summary statistic that is undefined, e.g., time to first spike is

undefined in the absence of spikes (Fig. 6c). When a summary statistic is undefined, we assign it a value that is substantially outside the range of the observed data (Appendix Sec. 6.14).

We ran TSNPE over six rounds (hyperparameters in Appendix Sec. 6.15). In each round, we ran 30k simulations, leading to a total of 180k simulations. After every round, we evaluated the expected coverage with SBCC. After the first round, the approximate posterior exhibited poor expected coverage (Fig. 6d top). Therefore, as suggested by Hermans et al. [2021], we used an ensemble of 10 neural density estimators to ensure that the approximate posterior is sufficiently broad (Fig. 6d bottom). Although the approximate posterior remains underconfident, the empirical expected coverage closely matches the confidence level for high-confidence levels, which is crucial for TSNPE (Sec. 3.3).

The TSNPE-posterior has several parameters with broad marginals, demonstrating that this model exhibits 'degeneracy', a widespread phenomenon in biological systems [Marder and Taylor, 2011] (Fig. 6e, Appendix Sec. 6.14 for parameter names). Other marginals are narrow, demonstrating that the model is sensitive to changes in these parameters. Posterior predictive samples closely match the observed data (Fig. 6f,g; more samples in Appendix Fig. 22). We emphasize that fitting such morphologically-detailed neuron models is a challenging and widespread problem in neuroscience, one for which commonly used methods [e.g., genetic algorithms, Druckmann et al., 2007, Van Geit et al., 2016] are often simulation-inefficient or do not estimate the full posterior distribution. We show promising results, suggesting that TSNPE could be applied to other complex single-neuron models.

In contrast, with 'out of the box' APT, none of the 10M approximate-posterior samples was within the prior bounds after the second round. This rendered a third round too computationally expensive. Overall, the results on the pyloric network model and on the multicompartment model demonstrate that TSNPE is an efficient and robust method that scales to complex and high-dimensional models that were inaccessible to the state-of-the-art method APT.

5 Discussion

We presented a new method to perform Bayesian inference in implicit models, which we call Truncated Sequential Neural Posterior Estimation (TSNPE). Like previous methods, TSNPE adaptively selects parameters to improve simulation-efficiency and allow posterior inference in complex models with many parameters. The key ingredient is that TSNPE samples parameters from a 'truncated' region of the prior, and thus overcomes instabilities of previous methods while maintaining simulation efficiency. In order to diagnose potential errors of TSNPE, we developed a coverage test that can be run quickly and at every round of TSNPE. TSNPE presents a new variant of SNPE which is at least as powerful as previous variants on benchmark tasks, but provides a powerful alternative which is able to solve inference problems on which the state-of-the-art method APT failed.

Related work TSNPE differs from automatic posterior transformation (APT, SNPE-C) in its proposal and its loss function: TSNPE uses a truncated prior as proposal, while APT can flexibly use any proposal, which, e.g., allows for more sophisticated active learning rules [Lueckmann et al., 2019, Järvenpää et al., 2019]. However, APT's flexibility requires a modification of NPE loss function, which can be an impediment to its usage in practice: First, the modification can lead to 'leakage', which can make it prohibitive to draw samples within prior bounds. Second, APT loss requires an explicit prior and thus, cannot be applied to models in which the prior can only be sampled [Ramesh et al., 2022]. Third, current formulations of APT cannot discard parameters leading to invalid simulations as the posterior mass would 'leak' into parameter regions which only produce invalid simulations (Appendix Sec. 6.5). It might be possible that these issues are resolved using a modified formulation of APT, e.g., by combining its atomic loss with additional loss terms, or preventing leakage by penalizing 'bad' parameters [Greenberg et al., 2019]. In cases in which leakage prevents application of APT (in particular, in high-dimensional problems), TSNPE provides an alternative.

Our method is inspired by previous work that introduced a mechanism to post-hoc correct samples obtained by an Approximate Bayesian Computation (ABC) algorithm [Blum and François, 2010], i.e., 'regression adjustment ABC'. Their method draws samples from a truncated region of the prior to avoid correction terms, but estimates the posterior density with ABC samples—rather than using a flexible neural density estimator—and estimates the support by training a dedicated support-vector machine. In addition, the method runs a single round of truncation and retraining, whereas we demonstrate that TSNPE can be robustly applied across 10 rounds.

Truncated proposals have also been proposed for neural ratio estimation [Truncated Marginal Neural Ratio Estimation (TMNRE) Miller et al., 2021]: TMNRE uses truncated proposals to efficiently infer selected posterior marginals while being amortised around the observation, allowing to test the coverage properties of the selected marginals, e.g., with SBC [Cook et al., 2006, Talts et al., 2018]. In addition, truncating based on the marginals allows TMRNE to sample from the truncated proposal without rejection or SIR sampling. In contrast, TSNPE aims at efficiently inferring the full posterior distribution by proposals that avoid the correction of SNPE loss function. Truncating the proposal based on the full posterior rather than on the marginals can lead to drastically narrower proposals: E.g., on the pyloric network problem, truncation based on posterior marginals rejects 20% of prior samples versus 99.94% rejection based on the posterior joint. In addition, while TMNRE uses the expected coverage to test the consistency of the posterior marginals, TSNPE can test the expected coverage of the full posterior distribution.

Possible failure modes The main failure mode of TSNPE will occur if the truncated proposal excludes significant portions of density mass of the true posterior (e.g., if the estimate misses posterior modes). In these cases, the learned approximate posterior will put systematically too little mass in the excluded regions. We recommend the use of diagnostic tools such as SBCC to identify such failures [Cook et al., 2006, Miller et al., 2021, Hermans et al., 2021, Rozet et al., 2021].

In addition, if the true posterior has unbounded support, any finite values of $\epsilon > 0$ will lead to a biased approximate posterior which puts too little weight in the posterior tails. In that case, and when running TSNPE across many rounds, the errors from each individual round could accumulate. Although we did not observe this bias to significantly affect the algorithm performance on several benchmark tasks, we cannot exclude the possibility of a substantial performance degradation when running TSNPE for a larger number of rounds ($\gg$10).

Finally, unlike SNPE methods that use the previous estimate of the posterior as the proposal distribution, our method requires a scheme to sample from a truncated proposal. If the sampling scheme is inaccurate (i.e., if it does not produce a proposal distribution that is proportional to the prior within the truncated region), the results of TSNPE will be biased. To avoid this, we recommend using rejection sampling by default and using SIR or sequential Monte-Carlo methods only if rejection sampling is too computationally expensive. For SIR, it is important to use a large oversampling factor K (e.g., $K = 1024$) and use diagnostic tools such as effective sample size (Appendix Sec. 6.12, Fig. 13).

Simulation-based coverage calibration In order to diagnose whether the approximate posterior is broader than the true posterior, we applied SBCC, a coverage test for TSNPE [Cook et al., 2006, Rozet et al., 2021]. SBCC evaluates the expected coverage of the approximate posterior without evaluating it on a grid [Dalmasso et al., 2020, Hermans et al., 2021] and, unlike diagnostic tools for methods based on learning the likelihood(-ratio), does not require MCMC runs for multiple observations $\mathbf{x}$ [Miller et al., 2021]. This allows SBCC to be run quickly and for models with many parameters. In addition, in contrast to diagnostic tools for likelihood-free inference with Approximate Bayesian Computation, SBCC does not require an additional step to estimate the density of approximate posterior samples [Prangle et al., 2014]. We note that, since SBCC is a variation of SBC [Cook et al., 2006, Talts et al., 2018], it only ensures that the HPR_ϵ is correct *on average* across observations, not for a particular observation. In principle, SBCC could be applied to other SNPE variants, although empirically the impact of arbitrary proposals on SBCC performance is currently unclear.

Conclusion Overall, TSNPE combines the simulation-efficiency of sequential neural posterior estimation with the robustness and coverage-tests of non-sequential methods. We demonstrated that it allows to scale neural posterior estimation to complex and high-dimensional scientific problems.

Acknowledgments and Disclosure of Funding

We thank Poornima Ramesh, Cornelius Schröder, Marcel Nonnenmacher, David Greenberg, and Jan-Matthis Lueckmann for discussions and feedback. We also thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting MD. This work was funded by the German Research Foundation (DFG; Germany's Excellence Strategy MLCoE – EXC number 2064/1 PN 390727645) and the German Federal Ministry of Education and Research (BMBF; Tübingen AI Center, FKZ: 01IS18039A).

References

M A Beaumont, J Cornuet, J Marin, and C P Robert. Adaptive approximate bayesian computation. *Biometrika*, 2009.

Mark A Beaumont, Wenyang Zhang, and David J Balding. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Michael GB Blum and Olivier François. Non-linear regression models for approximate bayesian computation. *Statistics and Computing*, 20(1):63–73, 2010.

Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

Frédéric Cérou and Arnaud Guyader. Adaptive multilevel splitting for rare event analysis. *Stochastic Analysis and Applications*, 25(2):417–443, 2007.

Samantha R Cook, Andrew Gelman, and Donald B Rubin. Validation of software for bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics*, 15(3):675–692, 2006.

Niccolò Dalmasso, Taylor Pospisil, Ann B Lee, Rafael Izbicki, Peter E Freeman, and Alex I Malz. Conditional density estimation tools in python and r with applications to photometric redshifts and likelihood-free cosmological inference. *Astronomy and Computing*, 30:100362, 2020.

Maximilian Dax, Stephen R Green, Jonathan Gair, Jakob H Macke, Alessandra Buonanno, and Bernhard Schölkopf. Real-time gravitational wave science with neural posterior estimation. *Physical review letters*, 127(24):241103, 2021.

Maximilian Dax, Stephen R Green, Jonathan Gair, Michael Deistler, Bernhard Schölkopf, and Jakob H. Macke. Group equivariant neural posterior estimation. In *International Conference on Learning Representations*, 2022.

Michael Deistler, Jakob H Macke, and Pedro J Gonçalves. Energy efficient network activity from disparate circuit parameters. *bioRxiv*, 2021.

Petar M Djuric, Jayesh H Kotecha, Jianqui Zhang, Yufei Huang, Tadesse Ghirmai, Mónica F Bugallo, and Joaquin Miguez. Particle filtering. *IEEE signal processing magazine*, 20(5):19–38, 2003.

Arnaud Doucet, Nando De Freitas, Neil James Gordon, et al. *Sequential Monte Carlo methods in practice*, volume 1. Springer, 2001.

Shaul Druckmann, Yoav Banitt, Albert A Gidon, Felix Schürmann, Henry Markram, and Idan Segev. A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data. *Frontiers in Neuroscience*, 1:1, 2007.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *Advances in neural information processing systems*, 32, 2019.

Conor Durkan, Iain Murray, and George Papamakarios. On contrastive learning for likelihood-free inference. In *International Conference on Machine Learning*, pages 2771–2781. PMLR, 2020.

Manuel Glöckler, Michael Deistler, and Jakob H. Macke. Variational methods for simulation-based inference. In *International Conference on Learning Representations*, 2022.

Pedro J Gonçalves, Jan-Matthis Lueckmann, Michael Deistler, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F Podlaski, Sara A Haddad, Tim P Vogels, et al. Training deep neural density estimators to identify mechanistic models of neural dynamics. *Elife*, 9:e56261, 2020.

David Greenberg, Marcel Nonnenmacher, and Jakob Macke. Automatic posterior transformation for likelihood-free inference. In *International Conference on Machine Learning*, pages 2404–2414. PMLR, 2019.

Sara Ann Haddad and Eve Marder. Recordings from the c. borealis stomatogastric nervous system at different temperatures in the decentralized condition. *URL https://doi.org/10.5281/zenodo*, 5139650, 2021.

Etay Hay, Sean Hill, Felix Schürmann, Henry Markram, and Idan Segev. Models of neocortical layer 5b pyramidal cells capturing a wide range of dendritic and perisomatic active properties. *PLoS computational biology*, 7(7):e1002107, 2011.

Joeri Hermans, Volodimir Begy, and Gilles Louppe. Likelihood-free mcmc with amortized approximate ratio estimators. In *International Conference on Machine Learning*, pages 4239–4248. PMLR, 2020.

Joeri Hermans, Arnaud Delaunoy, François Rozet, Antoine Wehenkel, and Gilles Louppe. Averting a crisis in simulation-based inference. *arXiv preprint arXiv:2110.06581*, 2021.

Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

Marko Järvenpää, Michael U Gutmann, Arijus Pleska, Aki Vehtari, and Pekka Marttinen. Efficient acquisition rules for model-based approximate bayesian computation. *Bayesian Analysis*, 14(2): 595–622, 2019.

William Ogilvy Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character*, 115(772):700–721, 1927.

Augustine Kong. A note on importance sampling using standardized weights. *University of Chicago, Dept. of Statistics, Tech. Rep*, 348, 1992.

Jan-Matthis Lueckmann, Pedro J Goncalves, Giacomo Bassetto, Kaan Öcal, Marcel Nonnenmacher, and Jakob H Macke. Flexible statistical inference for mechanistic models of neural dynamics. *Advances in neural information processing systems*, 30, 2017.

Jan-Matthis Lueckmann, Giacomo Bassetto, Theofanis Karaletsos, and Jakob H Macke. Likelihood-free inference with emulator networks. In *Symposium on Advances in Approximate Bayesian Inference*, pages 32–53. PMLR, 2019.

Jan-Matthis Lueckmann, Jan Boelts, David Greenberg, Pedro Goncalves, and Jakob Macke. Benchmarking simulation-based inference. In *International Conference on Artificial Intelligence and Statistics*, pages 343–351. PMLR, 2021.

Eve Marder and Adam L Taylor. Multiple models to capture the variability in biological neurons and networks. *Nature neuroscience*, 14(2):133–138, 2011.

Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, Selim Arsever, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.

Luca Martino, Víctor Elvira, and Francisco Louzada. Effective sample size for importance sampling based on discrepancy measures. *Signal Processing*, 131:386–401, 2017.

Benjamin Miller, Alex Cole, Patrick Forré, Gilles Louppe, and Christoph Weniger. Truncated marginal neural ratio estimation. *Advances in Neural Information Processing Systems*, 34, 2021.

Benjamin Kurt Miller, Alex Cole, Gilles Louppe, and Christoph Weniger. Simulation-efficient marginal posterior estimation with swyft: stop wasting your precious time. *arXiv preprint arXiv:2011.13951*, 2020.

Brooks Paige and Frank Wood. Inference networks for sequential monte carlo in graphical models. In *International Conference on Machine Learning*, pages 3040–3049. PMLR, 2016.

George Papamakarios and Iain Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. *Advances in neural information processing systems*, 29, 2016.

George Papamakarios, David Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 837–848. PMLR, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Dennis Prangle, Michael GB Blum, G Popovic, and SA Sisson. Diagnostic tools for approximate bayesian computation using the coverage property. *Australian & New Zealand Journal of Statistics*, 56(4):309–329, 2014.

Astrid A Prinz, Cyrus P Billimoria, and Eve Marder. Alternative to hand-tuning conductance-based models: construction and analysis of databases of model neurons. *Journal of neurophysiology*, 2003.

Astrid A Prinz, Dirk Bucher, and Eve Marder. Similar network activity from disparate circuit parameters. *Nature neuroscience*, 7(12):1345–1352, 2004.

Stefan T Radev, Ulf K Mertens, Andreas Voss, Lynton Ardizzone, and Ullrich Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

Srikanth Ramaswamy, Jean-Denis Courcol, Marwan Abdellah, Stanislaw R Adaszewski, Nicolas Antille, Selim Arsever, Guy Atenekeng, Ahmet Bilgili, Yury Brukau, Athanassia Chalimourda, et al. The neocortical microcircuit collaboration portal: a resource for rat somatosensory cortex. *Frontiers in neural circuits*, 9:44, 2015.

Poornima Ramesh, Jan-Matthis Lueckmann, Jan Boelts, Álvaro Tejero-Cantero, David S. Greenberg, Pedro J. Goncalves, and Jakob H. Macke. GATSBI: Generative adversarial training for simulation-based inference. In *International Conference on Learning Representations*, 2022.

François Rozet et al. Arbitrary marginal neural ratio estimation for likelihood-free inference. *Université de Liège, Liège, Belgique*, 2021.

Donald B Rubin. Using the sir algorithm to simulate posterior distributions. *Bayesian statistics*, 3: 395–402, 1988.

John Skilling. Nested sampling. In *Aip conference proceedings*, volume 735, pages 395–405. American Institute of Physics, 2004.

Sean Talts, Michael Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. Validating bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:1804.06788*, 2018.

Alvaro Tejero-Cantero, Jan Boelts, Michael Deistler, Jan-Matthis Lueckmann, Conor Durkan, Pedro J. Gonçalves, David S. Greenberg, and Jakob H. Macke. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52):2505, 2020. doi: 10.21105/joss.02505.

Owen Thomas, Ritabrata Dutta, Jukka Corander, Samuel Kaski, and Michael U Gutmann. Likelihood-free inference by ratio estimation. *Bayesian Analysis*, 17(1):1–31, 2022.

Werner Van Geit, Michael Gevaert, Giuseppe Chindemi, Christian Rössert, Jean-Denis Courcol, Eilif B Muller, Felix Schürmann, Idan Segev, and Henry Markram. Bluepyopt: leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. *Frontiers in neuroinformatics*, page 17, 2016.

Peter J Wangersky. Lotka-volterra population models. *Annual Review of Ecology and Systematics*, 9 (1):189–218, 1978.

Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466 (7310):1102–1104, 2010.

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019.

6 Appendix

6.1 Reproducibility statement

We used the configuration manager hydra to track the configuration and seeds of each run [Yadan, 2019]. All code to reproduce the results can be found at `https://github.com/mackelab/tsnpe_neurips`. We implemented TSNPE on top of the publicly accessible sbi toolbox [Tejero-Cantero et al., 2020]. All simulations and runs were performed on a high-performance computer. For each run, we used between 8 and 48 CPU cores.

6.2 Proof of convergence

Below, we prove that, for a given observation $\mathbf{x}_o$, the posterior distribution obtained with TSNPE $q_\phi(\boldsymbol{\theta}|\mathbf{x}_o)$ converges to the true posterior distribution $p(\boldsymbol{\theta}|\mathbf{x}_o)$ under the assumption that the HPR_ϵ of the approximate posterior at every TSNPE round covers the support (i.e., the HPR_ϵ for $\epsilon = 0$) of the true posterior given the observation $\mathbf{x}_o$. We call the support of the true posterior $\mathcal{M}$, i.e. $\mathcal{M} = \text{supp}(p(\boldsymbol{\theta}|\mathbf{x}_o))$. The proof proceeds in two steps: First, we derive the effective proposal distribution when pooling data from all rounds. Second, we show that, for such proposal distributions, the neural density estimator converges to the true posterior.

Deriving the proposal distribution We denote by $\tilde{p}^r(\boldsymbol{\theta})$ the proposal distribution from which $\boldsymbol{\theta}$ are drawn in round r. In the first round, we use the prior, i.e., $\tilde{p}^{r=1}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$. In later rounds, we sample from the prior but reject samples that lie outside of the HPR_ϵ of the approximate posterior given $\mathbf{x}_o$. The samples drawn in round r are thus drawn from

$$\tilde{p}^r(\boldsymbol{\theta}) = U^r(\boldsymbol{\theta})p(\boldsymbol{\theta})/Z^r, \tag{1}$$

where Z^r is the normalization constant and $U^r(\boldsymbol{\theta})$ is 1 on the HPR_ϵ of the approximate posterior at round r and zero otherwise. Given our assumption that the HPR_ϵ of the approximate posterior covers the support of the true posterior $\mathcal{M}$, $U^r(\boldsymbol{\theta})$ is 1 on the support of the true posterior $\mathcal{M}$.

When pooling simulations from all rounds, after R rounds, the parameters are sampled from a mixture of all proposal distributions:

$$\tilde{p}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{r=1}^{R}\tilde{p}^r(\boldsymbol{\theta}) = p(\boldsymbol{\theta}) \cdot \left(\frac{1}{N}\sum_{r=1}^{R}U^r(\boldsymbol{\theta})/Z^r\right) = p(\boldsymbol{\theta}) \cdot f(\boldsymbol{\theta}). \tag{2}$$

In this equation, we assumed that all rounds contain equally many simulations, but the proof can easily be extended to rounds with different numbers of simulations by adding weights to the above sum.

As can be seen above, the distribution $\tilde{p}(\boldsymbol{\theta})$ is the prior times a function $f(\boldsymbol{\theta})$ which is made up of several steps and whose steps are defined by the HPR_ϵ of the approximate posterior of every round (illustration in Appendix Fig. 23). Finally, under the assumption that all $U^r(\boldsymbol{\theta})$ are 1 on the support of the true posterior $\mathcal{M}$, we have, for any $\boldsymbol{\theta} \in \mathcal{M}$

$$f(\boldsymbol{\theta}) = \frac{1}{N}\sum_{r=1}^{R}U^r(\boldsymbol{\theta})/Z_1^r = \frac{1}{N}\sum_{r=1}^{R}1/Z_1^r = \text{constant} = c,$$

i.e., $f(\boldsymbol{\theta})$ is not a function of $\boldsymbol{\theta}$. Thus, for any $\boldsymbol{\theta} \in \mathcal{M}$, we have

$$\tilde{p}(\boldsymbol{\theta}) = p(\boldsymbol{\theta}) \cdot c \propto p(\boldsymbol{\theta})$$

We emphasise that this proportionality holds only for $\boldsymbol{\theta} \in \mathcal{M}$, but not necessarily for $\boldsymbol{\theta}$ outside of the support of the true posterior.

Training the neural density estimator Next, we show that, for a proposal distribution of the form derived in the paragraph above, the approximate posterior $q_\phi(\boldsymbol{\theta}|\mathbf{x}_o)$ for an observation $\mathbf{x}_o$ converges to the true posterior distribution $p(\boldsymbol{\theta}|\mathbf{x}_o)$.

TSNPE minimizes the following loss function:

$$\begin{aligned}\mathcal{L} &= -\frac{1}{N}\sum_i \log(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_i|\mathbf{x}_i)) \\ &\xrightarrow{N\to\infty} -\mathbb{E}_{p(\boldsymbol{\theta},\mathbf{x})}[\log(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}))] \\ &= -\mathbb{E}_{\tilde{p}(\boldsymbol{\theta})p(\mathbf{x}|\boldsymbol{\theta})}[\log(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}))].\end{aligned}$$

Plugging in the proposal distribution defined above:

$$\begin{aligned}\mathcal{L} &= -\iint f(\boldsymbol{\theta})p(\boldsymbol{\theta})p(\mathbf{x}|\boldsymbol{\theta})\log(q_\phi(\boldsymbol{\theta}|\mathbf{x}))\, d\boldsymbol{\theta}d\mathbf{x} \\ &= \int p(\mathbf{x})\int -f(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x})\log(q_\phi(\boldsymbol{\theta}|\mathbf{x}))\, d\boldsymbol{\theta}d\mathbf{x}.\end{aligned}$$

The term within the integral over $\boldsymbol{\theta}$ is proportional to the Kullback-Leibler-divergence between $f(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x})/Z$ (with $Z = \int f(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x})d\boldsymbol{\theta}$) and the approximate posterior $q_\phi(\boldsymbol{\theta}|\mathbf{x})$. Thus, $\mathcal{L}$ is minimized if and only if

$$q_\phi(\boldsymbol{\theta}|\mathbf{x}) \propto f(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x})$$

for all $\mathbf{x}$ within the the support of $p(\mathbf{x})$ [Papamakarios and Murray, 2016].

This means that, for arbitrary $\mathbf{x} \in \text{supp}(p(\mathbf{x}))$, $q_\phi(\boldsymbol{\theta}|\mathbf{x})$ will not converge to the true posterior $p(\boldsymbol{\theta}|\mathbf{x})$, but to $f(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x})/Z$. However, for the observed data $\mathbf{x} = \mathbf{x}_o$, we have:

$$q_\phi(\boldsymbol{\theta}|\mathbf{x}_o) \propto f(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x}_o) = \begin{cases} c \cdot p(\boldsymbol{\theta}|\mathbf{x}_o) & \text{if } \boldsymbol{\theta} \in \mathcal{M} \\ f(\boldsymbol{\theta}) \cdot 0 & \text{else} \end{cases}$$

The first case follows from the fact that $f(\boldsymbol{\theta})$ is constant on the support of the true posterior. The second case follows because the true posterior has zero probability density for $\boldsymbol{\theta}$ outside of its own support $\mathcal{M}$. Thus:

$$q_\phi(\boldsymbol{\theta}|\mathbf{x}_o) \propto p(\boldsymbol{\theta}|\mathbf{x}_o).$$

Since $q_\phi(\boldsymbol{\theta}|\mathbf{x}_o)$ is a (conditional) normalizing flow, it is normalized and, thus:

$$q_\phi(\boldsymbol{\theta}|\mathbf{x}_o) = p(\boldsymbol{\theta}|\mathbf{x}_o).$$

6.3 Simulation-based coverage calibration (SBCC)

The algorithm for computing the coverage (SBCC) is shown in Alg. 2.

Algorithm 2: Simulation-based coverage calibration (SBCC)

Inputs: proposal $\tilde{p}(\boldsymbol{\theta})$, approximate posterior $q_{\boldsymbol{\phi}}$ trained on $\boldsymbol{\theta} \sim \tilde{p}(\boldsymbol{\theta})$, number of simulations M, number of posterior samples per simulation P. Database of empirical coverages $\mathcal{E}$, initialized as the empty set.
Outputs: Coverage for different confidence levels $1 - \alpha$.

```
for i ∈ [1, ..., M] do
    θ*_i ~ p̃(θ)
    simulate x*_i ~ p(x|θ*_i)
    l*_i = log(q_φ(θ*_i|x*_i)) ;                    // compute log-prob of ground-truth
    c = 0
    for j ∈ [1, ..., P] do
        θ_j ~ q_φ(θ|x*_i)
        l_j = log(q_φ(θ_j|x*_i)) ;                  // compute log-probs of posterior samples
        if l_j > l*_i then
            c = c + 1
        e = c/P ;   // fraction of posterior samples whose log-prob is larger
         than ground-truth log-prob
        add e to E
Plot CDF of E
```

Algorithm 3: Obtaining the HPR_ϵ of the approximate posterior and sampling the truncated proposal with rejection sampling

Inputs: Prior $p(\boldsymbol{\theta})$, Approximate posterior $q_{\boldsymbol{\phi}}$, observation $\mathbf{x}_o$, number of posterior samples M, ϵ that defines the HPR, number of desired samples from $\tilde{p}(\theta)$ N. Database of log-probabilities $\mathcal{P}$, initialized as an empty set.
Outputs: Samples $\mathcal{S}$ from truncated proposal $\tilde{p}(\boldsymbol{\theta})$.
for $i \in [1, ..., M]$ **do**
  $\boldsymbol{\theta}_i \sim q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o)$
  $l_i = \log(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_i|\mathbf{x}_o)$
  add l_i to $\mathcal{P}$
$\boldsymbol{\kappa}$ = quantile$_\epsilon(\mathcal{P})$; `// threshold for HPR`$_\epsilon$

`// sample truncated proposal with rejection sampling`
while $s < N$ **do**
  $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$
  $l = \log(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o))$
  if $l > \boldsymbol{\kappa}$ **then**
    add $\boldsymbol{\theta}$ to $\mathcal{S}$
    s += 1

6.4 Algorithm description for sampling from the HPR of the approximate posterior

The algorithm for sampling from the HPR_ϵ of the approximate posterior is shown in Alg. 3.

6.5 Alleviating issues of APT

We compared TSNPE to Automatic Posterior Transformation (APT) [Greenberg et al., 2019]. In this section, we briefly review how APT works, and why 'leakage' occurs, and how we attempted to improve it.

APT review Broadly, APT exists in two versions. Its first version can be applied only if the density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ is a mixture of Gaussians, the proposal $\tilde{p}(\boldsymbol{\theta})$ is a mixture of Gaussians, and the prior $p(\boldsymbol{\theta})$ is either uniform or Gaussian. We did not compare TSNPE to APT in this form because we wanted to use more expressive density estimators for $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$. The second version of APT, known as atomic APT, allows to use any density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$, any proposal $\tilde{p}(\boldsymbol{\theta})$ and any explicit prior $p(\boldsymbol{\theta})$. One must be able to evaluate the density estimator and the prior, but the proposal can be implicit (i.e., without closed form density). Atomic APT minimizes the loss:

$$\mathcal{L}_{\boldsymbol{\phi}} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_i|\mathbf{x}_i)/p(\boldsymbol{\theta}_i)}{q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_i|\mathbf{x}_i)/p(\boldsymbol{\theta}_i) + \sum_{j=1\ldots A-1} q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_j|\mathbf{x}_i)/p(\boldsymbol{\theta}_j)}$$

with number of atoms A. In this loss, $\boldsymbol{\theta}_j$ and $\boldsymbol{\theta}_i$ can be sampled from *any* proposal distribution. $\mathbf{x}_i$ is sampled from $p(\mathbf{x}|\boldsymbol{\theta}_i)$.

Leakage issue Notice that the above loss is the same for $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ and for $c \cdot q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$. In other words, the approximate posterior has to be correct only up to a proportionality constant [Greenberg et al., 2019, Durkan et al., 2020]. Thus, an approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ which is $c \cdot p(\boldsymbol{\theta}|\mathbf{x})$ will have a minimal loss. Since the dataset on which $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ is trained contains no $\boldsymbol{\theta}$ that lies outside of the prior bounds, the approximate posterior can be anything outside of the prior bounds without affecting the value of the loss function. This is what is called 'leakage', and which has been pointed out as a potential problem both in the original APT paper Greenberg et al. [2019] and work studying the relationship of APT with contrastive learning approaches Durkan et al. [2020]: While the approximate posterior might be proportional to the true posterior within the bounds of the prior $p(\boldsymbol{\theta})$, it can put significant mass outside of the prior bounds (because the loss does not penalize this behaviour).

Transforming the parameter space We tried to fix the leakage issue by appending a transformation such that the density estimator has constrained support. This requires that the bounds of the parameter

space are known and that such a transformation can be implemented. If this fix can be applied, the leakage will be zero by definition. In all our experiments, such a transformation substantially helped with the leakage problem.

Leakage into regions of no training data However, even when possible, such transformation does not completely prevent 'leakage': When inspecting the loss of APT, we see that the approximate posterior can put mass into any region of the parameter space in which no parameter sets $\boldsymbol{\theta}$ in the training data lie. For example, if the prior distribution is standard Gaussian and one trains APT on 100 parameter sets sampled from the prior, the training dataset will unlikely contain values that are smaller than -3 or larger than 3. Therefore, APT can 'leak' into the regions below -3 or above 3 and still have an optimal loss. While, in the limit of infinite data, APT would correct its 'leakage' as soon as parameters from these regions are used as training data, given a finite number of simulations, the approximate posterior can 'leak' into new regions which have not been explored (yet). The (potential) problem of leakage does not affect all applications equally: High-dimensional parameter spaces suffer more from this behavior than low-dimensional ones, as there are many regions into which the mass of the approximate posterior can 'leak'. This behaviour is illustrated in Appendix Fig. 17.

Leakage because of invalid data Another way in which leakage can occur is if the simulator produces invalid data (e.g. NaN or infinity). Often, such invalid simulations are discarded from the training dataset [Lueckmann et al., 2017] and the approximate posterior is trained only on samples that produce valid outputs. For example, assume that a small region of the prior always produces invalid simulations. In this case, the approximate posterior will never be trained on simulations from this parameter region and APT can 'leak' into this region. Thus, the approximate posterior might contain significant mass in parameter regions that produce invalid simulations.

Explicit recommendations for running APT We will now give explicit recommendations for running APT. These modification greatly improved the performance of APT in our experiments, but were not able to avoid the failure of the algorithm on challenging real-world problems such as the pyloric network (Fig. 5).

1. For priors with bounded supports: If possible, transform the parameter space into unbounded space.
2. Do not discard invalid simulations. Instead, replace invalid entries (such as NaN) with a substantially different value than the observed data and train on all available simulations.
3. Check if the posterior contains a lot of mass in regions with very low prior probability. If this is the case, it can hint at a failure of APT through leakage into regions of no training data.
4. If you transformed the parameter space, check if many posterior samples lie very close to the bounds. Again, this can hint at a failure of APT through leakage into regions of no training data.

6.6 Relation between simulation-based calibration and our diagnostic

As discussed above, our method is closely related to simulation-based calibration (SBC) [Cook et al., 2006, Talts et al., 2018]. Briefly, SBC samples $\boldsymbol{\theta}^*$ from the prior, samples the likelihood $\mathbf{x}^* \sim p(\mathbf{x}|\boldsymbol{\theta}^*)$ and then draws samples from the posterior $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}|\mathbf{x}^*)$ (e.g., with MCMC). It then projects the (potentially high-dimensional) parameters $\boldsymbol{\theta}_i$ into a one-dimensional space $T(\boldsymbol{\theta}_i)$. Often, this projection is the 1D-marginal distribution of parameters [Carpenter et al., 2017]. It then ranks $T(\boldsymbol{\theta}^*)$ under all posterior samples $T(\boldsymbol{\theta}_{1\ldots N})$. Repeated across several prior samples, the distribution of ranks should be uniform. For high-dimensional parameter spaces, the marginal distribution of each parameter is checked independently.

Notably, other projections into a one-dimensional space are possible. Below, we explain that our method is identical to running SBC with projection $T : \boldsymbol{\theta} \to q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$ and with posterior samples exactly following $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$.

In SBC and in our diagnostic method, samples are drawn from the prior $\boldsymbol{\theta}^* \sim p(\boldsymbol{\theta})$ and simulated $\mathbf{x}^* \sim p(\mathbf{x}|\boldsymbol{\theta})$. In our diagnostic method, as well as in SBC with projection being the log-probability

of the approximate posterior, one then samples the posterior to obtain $\boldsymbol{\theta}_i$ and evaluates the log-probability of all samples, i.e., $l_i = T(\boldsymbol{\theta}_i) = \log(q(\boldsymbol{\theta}_i|\mathbf{x}^*)$ as well as of the initial parameter set $l^* = \log(q(\boldsymbol{\theta}^*|\mathbf{x}^*)$. SBC then ranks l^* under all l_i, which is equivalent to computing its quantile (as in our method). In our diagnostic tool, one then evaluates whether this quantile is above or below several confidence levels (evaluation on a 1D evenly spaced grid, same as rank binning in SBC). This generates a step-function with the step occuring at the quantile of l^*. This step function is the cumulative distribution function of a dirac at the quantile of l^*. Therefore, repeated across several $\boldsymbol{\theta}^*$, our coverage plots (e.g., Fig 3c) correspond to the cumulative distribution function of the histograms generated by SBC (with projection $T : \boldsymbol{\theta} \to \log(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}))$ and posterior samples exactly following $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x})$).

6.7 SBCC in a multi-round setting

We run our diagnostic tool after every round of training. If one trains only on simulations that were run in the most recent round, SBCC can be run as in the first round. However, if one wishes to train on simulations from all rounds, then the deep neural density estimator converges to:

$$q_\phi(\boldsymbol{\theta}|\mathbf{x}) \propto f(\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{x}).$$

Proof in Sec. 6.2. As is described in Sec. 6.2, this means that $q_\phi(\boldsymbol{\theta}|\mathbf{x})$ will not converge to the true posterior for arbitrary $\mathbf{x}$, but only for the observation $\mathbf{x}_o$.

This poses a problem for SBCC: As described in Alg. 2, SBCC measures whether the coverage is correct (on average) for many $\mathbf{x}$ generated by the proposal distribution. Since the loss employed by TSNPE only ensures convergence for $\mathbf{x}_o$, it will by construction not provide correct results for other $\mathbf{x}$.

This issue can be solved in two ways:

1. When running SBCC, instead of drawing samples from the proposal prior of the most recent round $\tilde{p}^r(\boldsymbol{\theta})$, one can draw samples $\boldsymbol{\theta}^*$ from $\tilde{p}(\boldsymbol{\theta})$, i.e., the distribution that emerges from pooling data from all rounds (notation as in Sec. 6.2 and Alg. 2). This is the method described in Appendix Alg. 2.
2. One can truncate the approximate posteriors $q_\phi(\boldsymbol{\theta}|\mathbf{x}^*)$ while running SBCC (see Alg. 2 for notation). With this strategy, when running SBCC in round r, we draw parameters from $\boldsymbol{\theta}^* \sim \tilde{p}^r(\boldsymbol{\theta})$ (the truncated proposal from round r), simulate them $\mathbf{x}^* \sim p(\mathbf{x}|\boldsymbol{\theta}^*)$, sample from the posterior $\boldsymbol{\theta}_i \sim q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}^*)$, reject samples that lie outside of $\text{HPR}_\epsilon(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_i|\mathbf{x}_o))$, and then continue as described in Alg. 2. Strategy 2 only ensures that the posterior regions which lie within $\text{HPR}_\epsilon(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}_o))$ are well-calibrated. It does not ensure that the full posterior $(q_{\boldsymbol{\phi}}(\boldsymbol{\theta}|\mathbf{x}^*)$ with $\mathbf{x}^* \sim p(\mathbf{x}^*|\boldsymbol{\theta}^*)$ and $\boldsymbol{\theta}^* \sim \tilde{p}^r(\boldsymbol{\theta}))$ is well-calibrated.

6.8 Toy model

The toy model shown in Fig. 1 is given by a uniform prior within $[-2, -1]$ and $[1, 2]$. The simulator is $x \sim \theta^2 + \epsilon$, where ϵ is a Gaussian distribution with mean zero and standard deviation 0.2. We ran APT [Greenberg et al., 2019] and TSNPE for 5 rounds with 500 simulations per round. For APT, all hyperparameters are the default values from the sbi package [Tejero-Cantero et al., 2020], but we used a neural spline flow (NSF) for both APT and TSNPE [Durkan et al., 2019].

6.9 Benchmark tasks

Below, we briefly describe the benchmark tasks. For details, please see Lueckmann et al. [2021].

Gaussian linear: 10 parameters which are the mean of a Gaussian model. The prior is Gaussian, resulting in a Gaussian posterior.

Bernoulli GLM: Generalized linear model with Bernoulli observations. Inference is performed on 10-dimensional sufficient summary statistics of the originally 100 dimensional raw data. The resulting posterior is 10-dimensional, unimodal, and concave.

Lotka Volterra: A traditional model in ecology [Wangersky, 1978], which describes a predator-prey interaction between species, illustrating a task with complex likelihood and unimodal posterior.

SLCP: A task introduced by Papamakarios et al. [2019] with a simple likelihood and complex posterior. The prior is uniform, the likelihood has Gaussian noise but is non-linearly related to the parameters, resulting in a posterior with four symmetrical modes.

Two moons: This model has two parameters with a uniform prior. The simulator is non-linear, generating a posterior with both local and global (bimodal) structure [Greenberg et al., 2019].

SIR: Epidemiological model with two parameters and ten summary statistics [Kermack and McKendrick, 1927].

6.10 Errors due to truncation

As described in Appendix Sec. 6.2, the approximate posterior converges to the true posterior if the truncated proposal covers the support of the true posterior. The truncated support is defined as the high probability region that contains 1-ϵ of mass of the approximate posterior (HPR$_\epsilon$). For $\epsilon > 0$, the HPR$_\epsilon$ will likely not be a superset of the support of the true posterior and hence, there will be errors in posterior approximation. In this section, we discuss the effect of these errors on inference accuracy.

When the value of ϵ is chosen too large, the tails of the approximate posterior are excluded from the truncated proposal. In the following training round, the approximate posterior converges to a distribution that is correct, up to proportionality, within the HPR$_\epsilon$ of the previous approximate posterior, but that underestimates the tails of the posterior distribution. We demonstrate this behavior in Appendix Fig. 11 for a linear Gaussian simulator, uniform prior, 50k simulations per round, and a neural spline flow with 20 bins as neural density estimator. After round 1, the approximate posterior closely matches the true posterior (Fig. 11a). When using a large ϵ, e.g., $\epsilon = 0.1$, the proposal for the second round is narrower than the true posterior and, thus, the proposal obtained by pooling data from both rounds is not constant on the support of the true posterior (Fig. 11b, blue). This leads to the approximate posterior underestimating the tails of the true posterior (Fig. 11b, purple). When using a smaller ϵ, e.g., $\epsilon = 0.01$, the errors induced by truncation become small and inference errors are mostly due to finite data and imperfect convergence of the neural network (Fig. 11c). We note that, throughout our study, we evaluated $\epsilon = \{10^{-3}, 10^{-4}, 10^{-5}\}$, i.e., values that are at least one order of magnitude smaller than $\epsilon = 0.01$.

Overall, this analysis demonstrates that the truncation performed by TSNPE can negatively impact inference quality in the tails of the posterior distribution. We, thus, do not recommend TSNPE in scenarios in which users are particularly interested in the tails of the posterior. In all our benchmark tasks, however, we did not find that the truncation negatively impacted inference quality as measured by C2ST accuracy (Fig. 4, Appendix Fig. 8, Fig. 8). This indicates that, for many (real-world) tasks, the errors due to truncation are outweighed by errors due to finite simulation budgets or imperfect convergence of neural network training.

6.11 Computational cost of rejection sampling and SIR

In this section, we quantify the computational costs of rejection sampling and sampling-importance resampling (SIR). The computational cost of both sampling methods comprises the computational cost of sampling and evaluating the approximate posterior. On an AMD Ryzen Threadripper 1920X 12-Core Processor, drawing (or evaluating) 100k samples takes approximately 10 seconds. On a GeForce RTX 2080 GPU, drawing (or evaluating) 100k samples takes approximately 0.17 seconds. Thus, in SIR (with an oversampling factor $K = 1024$), one can draw (or evaluate) 100 samples from the truncated proposal in 0.17 seconds on a GPU (versus 10 seconds on a CPU). For most real-world simulators, this constitutes a small fraction of the compute time required to simulate the model: e.g., for the multicompartment model, a single simulation takes approximately 30 seconds and, thus, SIR sampling from the truncated support takes up only 0.012% of the total compute time with a GPU. For rejection sampling, the time required to draw samples from the truncated support depends on the rejection rate. However, as long as the acceptance rate is above 0.001%, the cost of rejection sampling is still small compared to the cost of running the simulator.

6.12 Accuracy of SIR

Here, we investigate the (potential) error induced by using sampling-importance resampling (SIR). SIR is an approximate sampling technique and does not produce exact samples for finite K. This

raises the question of how strongly the errors induced by SIR influence the results of TSNPE. In order to investigate this, we performed three analyses: 1) We ran all benchmarking tasks with SIR and $K = 1024$ and compared the results to rejection sampling. Across all benchmark tasks, the performance of TSNPE with SIR matches the performance of TSNPE with rejection sampling (Appendix Fig. 10). 2) In a simple 1D toy model, we investigated how closely the samples produced by SIR match the samples produced by rejection sampling. As can be seen in Appendix Fig. 12, the distribution of SIR samples is quite different from rejection sampling for $K = 16$ and $K = 64$. However, for $K = 1024$, the distribution of samples from SIR very closely matches the distribution of rejection samples. 3) Finally, we investigated the performance of SIR by inspecting the effective sample size (ESS), which we computed as

$$\text{ESS} = 1/\sum_{i}^{K} w_i^2$$

with w_i being the normalized importance weights [Kong, 1992, Djuric et al., 2003, Martino et al., 2017]. For $K = 1024$, across all benchmark tasks, the ESS was on average 25.154 and was never below 2.547, i.e. it was always significantly higher than 1 (the number of resampled samples). All of these results indicate that SIR is expected to be a useful and robust sampling method for TSNPE.

6.13 Pyloric network model

For the pyloric network model, we used the same prior, simulator, and summary statistics as previous work [Gonçalves et al., 2020, Deistler et al., 2021, Glöckler et al., 2022]. The model has a total of 31 parameters and 18 summary statistics. We replaced invalid summary statistics with a value that is 2 standard deviations (of prior predictives) below the observation. The experimental data [Haddad and Marder, 2021] is also the same as used in these previous works.

6.14 Multicompartment model of a single neuron

We performed Bayesian inference in a complex model of single-neuron dynamics. The model is the same as used in Van Geit et al. [2016]. We added observation noise with standard deviations taken from previously published measurements [Hay et al., 2011]. The prior is a uniform distribution within the same bounds as previously used [Van Geit et al., 2016]. The parameters are shown in Table 1. The summary statistics are also the same as in Van Geit et al. [2016].

We replaced NaN values by the minimal value among prior samples minus two standard deviations of prior samples. Several summary statistics had heavy tailed distributions, which led to very high standard deviations. For these summary statistics, we picked the replacement value by hand. The final values are shown in Table 2.

6.15 Choices of hyperparameters

For the results on the benchmark tasks, we picked the same hyperparameters for TSNPE as those that were used in Lueckmann et al. [2021] for APT (called SNPE in Lueckmann et al. [2021]).

For both neuroscience tasks, we used a neural spline slow (NSF) as density estimator [Durkan et al., 2019]. The hyperparameters of the NSF are the defaults from the 'sbi' package [Tejero-Cantero et al., 2020]. On both of these tasks, we used $\epsilon = 10^{-3}$. All other hyperparameters are the defaults from the 'sbi' package with one exception: We used a batchsize of 500 (instead of the default value 50).

For the pyloric network task, we ran APT with 2 atoms. We reduced the number of atoms from the default value in the 'sbi' package (10 atoms) because a larger number of atoms increased training time. When using 10 atoms, the training time exceeded the simulation time of the model on the pyloric network task. With 2 atoms, the training time of APT was comparable to the training time of TSNPE (albeit still a bit higher). We implemented the transformation of the parameter space with the pytorch method 'biject_to()' [Paszke et al., 2019]. For TSNPE, we initially sampled from the truncated proposal with rejection sampling. After the eighth round of training, the rejection rate became exceedingly high and we switched sampling importance resampling (SIR, with $K = 1024$ see Sec. 3.2).

For the multicompartment model of single-neuron dynamics, we initially sampled parameters from the truncated proposal with rejection sampling and switched to SIR after the third round. For this

Index	Parameter	Ground truth
θ_1	gnats2_tbar_nats2_t_apical	0.026145
θ_2	gskv3_1bar_skv3_1_apical	0.004226
θ_3	gimbar_im_apical	0.000143
θ_4	gnata_tbar_nata_t_axonal	3.137968
θ_5	gk_tstbar_k_tst_axonal	0.089259
θ_6	gamma_cadynamics_e2_axonal	0.00291
θ_7	gnap_et2bar_nap_et2_axonal	0.006827
θ_8	gsk_e2bar_sk_e2_axonal	0.007104
θ_9	gca_hvabar_ca_hva_axonal	0.00099
θ_{10}	gk_pstbar_k_pst_axonal	0.973538
θ_{11}	gskv3_1bar_skv3_1_axonal	1.021945
θ_{12}	decay_cadynamics_e2_axonal	287.19873
θ_{13}	gca_lvastbar_ca_lvast_axonal	0.008752
θ_{14}	gamma_cadynamics_e2_somatic	0.000609
θ_{15}	gskv3_1bar_skv3_1_somatic	0.303472
θ_{16}	gsk_e2bar_sk_e2_somatic	0.008407
θ_{17}	gca_hvabar_ca_hva_somatic	0.000994
θ_{18}	gnats2_tbar_nats2_t_somatic	0.983955
θ_{19}	decay_cadynamics_e2_somatic	210.48529
θ_{20}	gca_lvastbar_ca_lvast_somatic	0.000333

Table 1: L5PC parameters.

task, in the first round of training, we used an ensemble of 10 neural networks. From the second round onward we used only a single neural network. For all other runs (toy example, benchmark, pyloric network), we did not use ensembles but always trained only a single network.

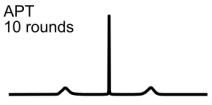


Figure 7: **APT performance after 10 rounds.** Same setup as in Fig. 1, but after running APT for 10 rounds. Leakage gets worse when additional rounds are run.

Summary statistic	Observation	Replacement value
step1_soma_ahp_depth_abs	-62.1358	-110.974
step1_soma_ahp_depth_abs_slow	-62.2882	-151.52
step1_soma_ahp_slow_time	0.140599	-0.8473
step1_soma_ap_height	28.43591	-33.959
step1_soma_ap_width	0.67857	-2.132
step1_soma_isi_cv	0.03328	-1.202
step1_soma_adaptation_index2	-0.0039499	-0.3790
step1_soma_doublet_isi	67.00	-1.699
step1_soma_mean_frequency	7.106	-52.343
step1_soma_time_to_first_spike	33.3000	-719.1
step2_soma_ahp_depth_abs	-60.6933	-110.974
step2_soma_ahp_depth_abs_slow	-60.8186	-151.066
step2_soma_ahp_slow_time	0.1496	-0.8481
step2_soma_ap_height	26.5820	-33.958
step2_soma_ap_width	0.67058	-2.1747
step2_soma_isi_cv	0.03598	-1.0649
step2_soma_adaptation_index2	-0.001467	-0.47684
step2_soma_doublet_isi	44.600	-1.6999
step2_soma_mean_frequency	8.8444	-67.842
step2_soma_time_to_first_spike	23.000	-719.1
step3_soma_ahp_depth_abs	-56.759	-110.744
step3_soma_ahp_depth_abs_slow	-55.903	-149.126
step3_soma_ahp_slow_time	0.2168	-0.83586
step3_soma_ap_height	16.968	-34.01
step3_soma_ap_width	0.5968	-2.6932
step3_soma_isi_cv	0.09933	-1.1164
step3_soma_adaptation_index2	0.007206	-0.5326
step3_soma_doublet_isi	21.100	-1.699
step3_soma_mean_frequency	16.086	-125.30
step3_soma_time_to_first_spike	10.600	-719.1
bap_dend1_ap_amplitude_from_voltagebase	53.267	-60.701
bap_dend2_ap_amplitude_from_voltagebase	30.592	-31.779
bap_soma_ap_height	37.519	-33.95
bap_soma_ap_width	0.800	-1.583
bap_soma_spikecount	1.0	-0.8855

Table 2: L5PC summary statistics.

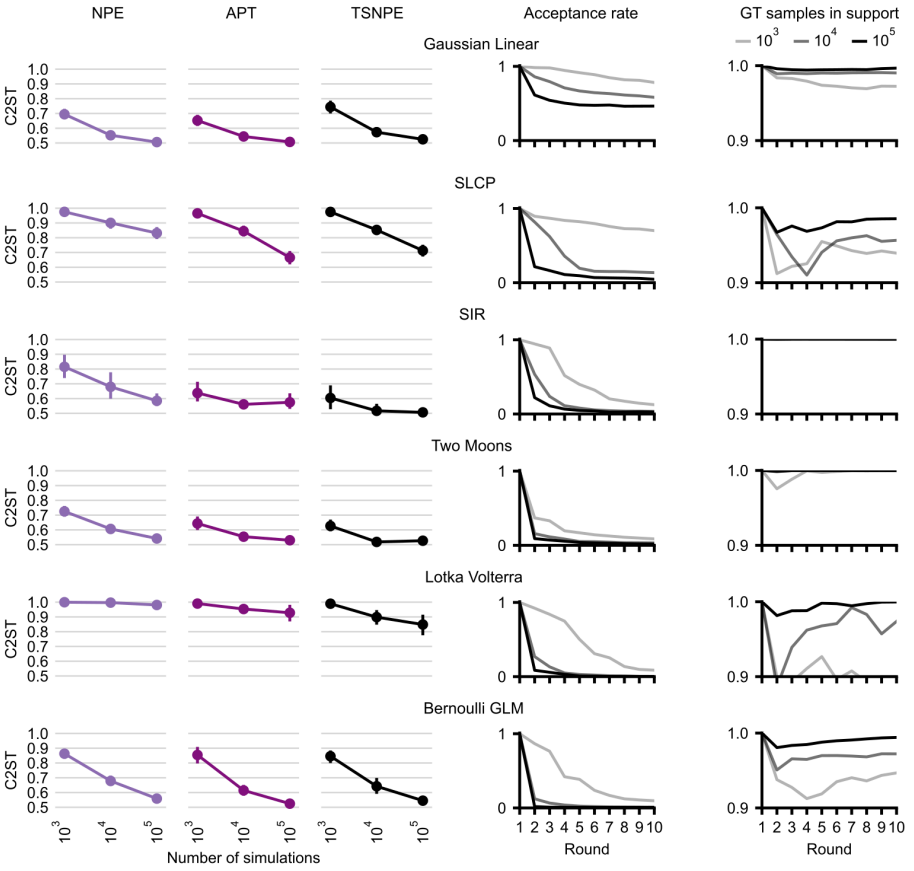


Figure 8: **Benchmark results for a less conservative threshold.** All hyperparameters are the same as in Fig. 4. The only difference is that we used $\epsilon = 10^{-3}$ (compared to 10^{-4} in Fig. 4).

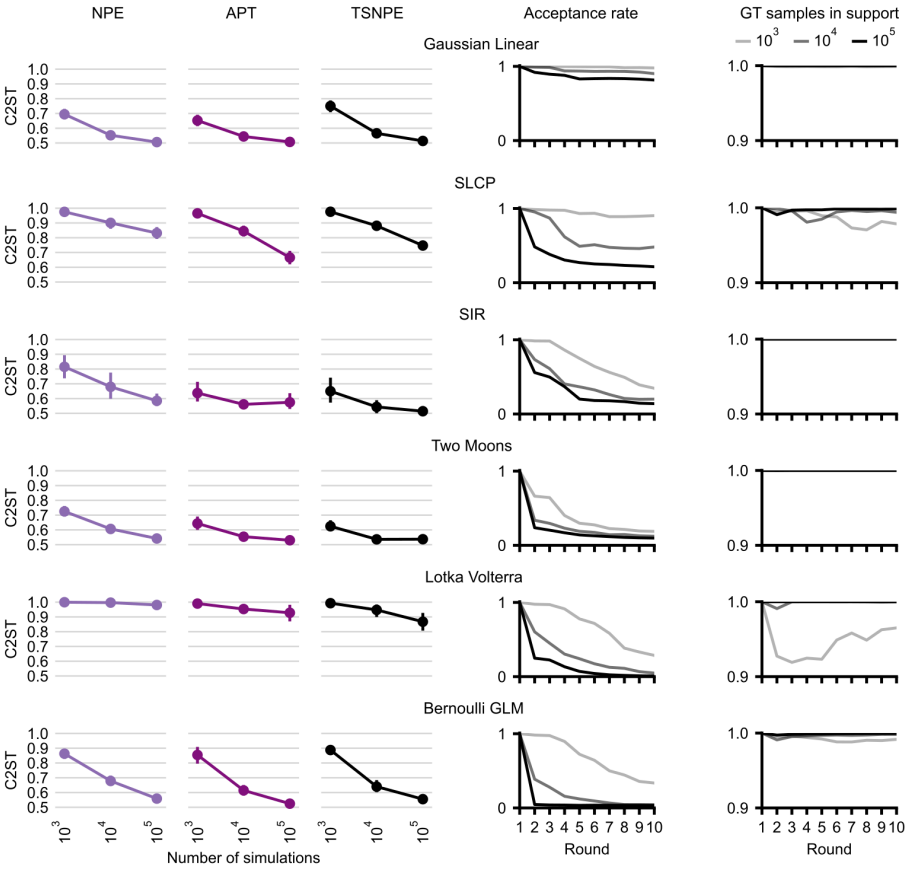


Figure 9: **Benchmark results for a more conservative threshold.** All hyperparameters are the same as in Fig. 4. The only difference is that we used $\epsilon = 10^{-5}$ (compared to 10^{-4} in Fig. 4).

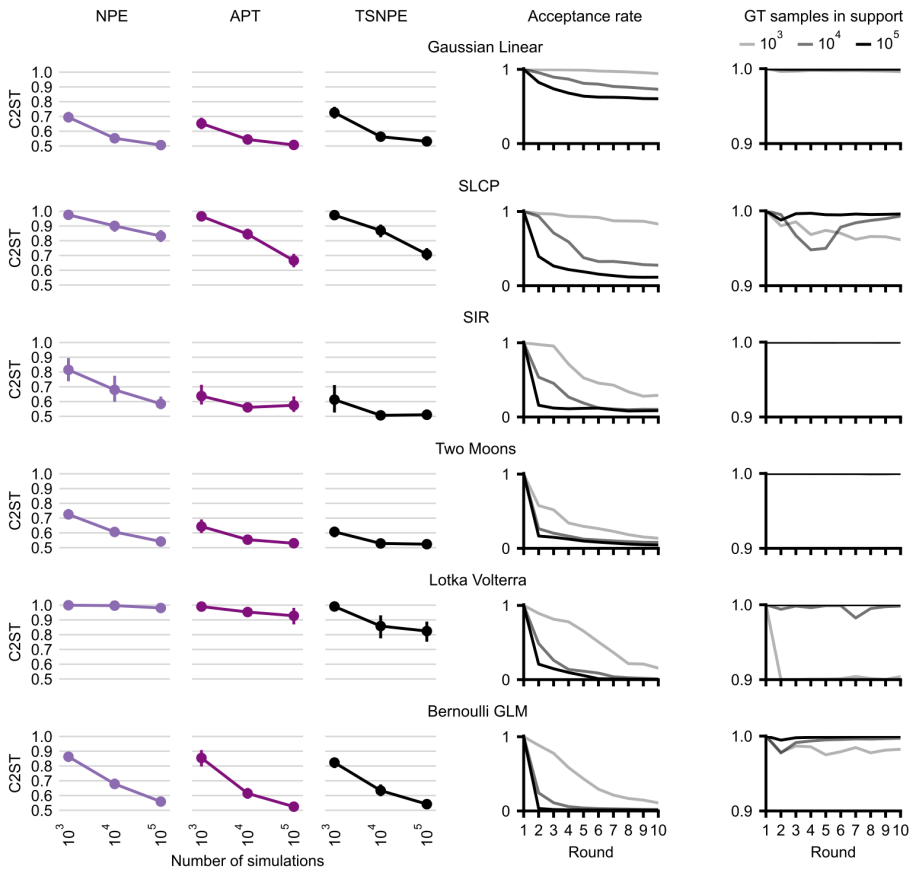


Figure 10: **Benchmark results for sampling importance resampling (SIR).** All hyperparameters are the same as in Fig. 4 ($\epsilon = 10^{-4}$) but we use SIR to sample from the truncated proposal (instead of rejection sampling in Fig. 4).

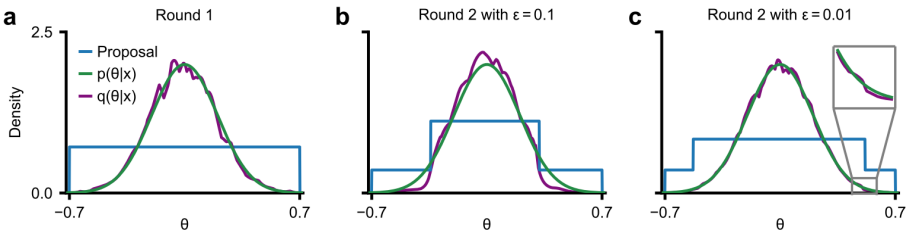


Figure 11: **Errors induced by truncation.** Inference in a linear Gaussian toy model with uniform prior. (a) The approximate posterior after round 1 closely matches the true posterior. (b) For a large truncation value $\epsilon = 0.1$, the approximate posterior after round 2 systematically underestimates the tails of the true posterior distribution. (c) For a smaller truncation value $\epsilon = 0.01$, the error induced by truncation is small.

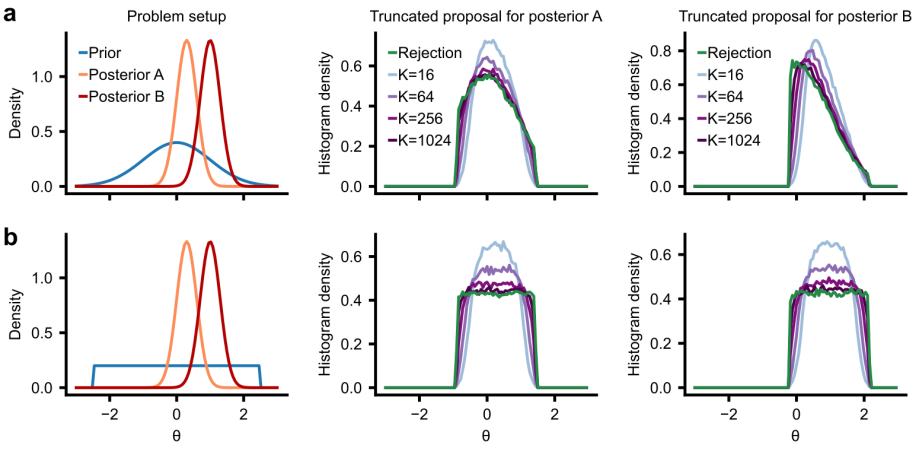


Figure 12: **Comparison of truncated proposal between rejection and importance sampling.** (a) Left: Gaussian prior as well as two posteriors. Middle: Density of 100k samples from the truncated proposal for posterior A. Green is rejection sampling, purple shaded colors are SIR with different oversampling factors K. Right: Same as middle, but for posterior B. (b) Same as panel a, but for a uniform prior.

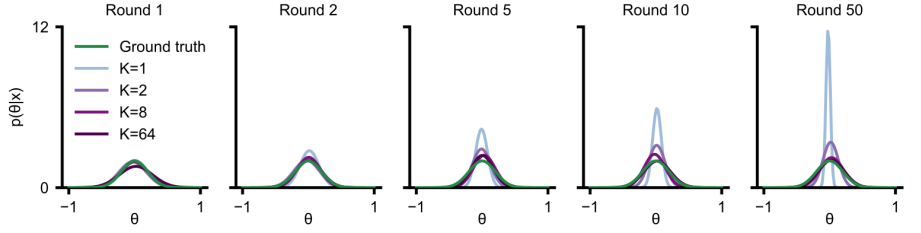


Figure 13: **Poor hyperparameter choices can lead SIR to diverge.** We applied TSNPE to a model with uniform prior (in [-1, 1]) and a linear Gaussian simulator. In each round, we ran 500 simulations, trained only on data from the most recent round, and used a Gaussian approximate posterior. We sampled from the truncated proposal with SIR (with different oversampling factors K). As more rounds are being run, the TSNPE approximate posterior can become too narrow for small K. Larger values of K are robust across 50 rounds.

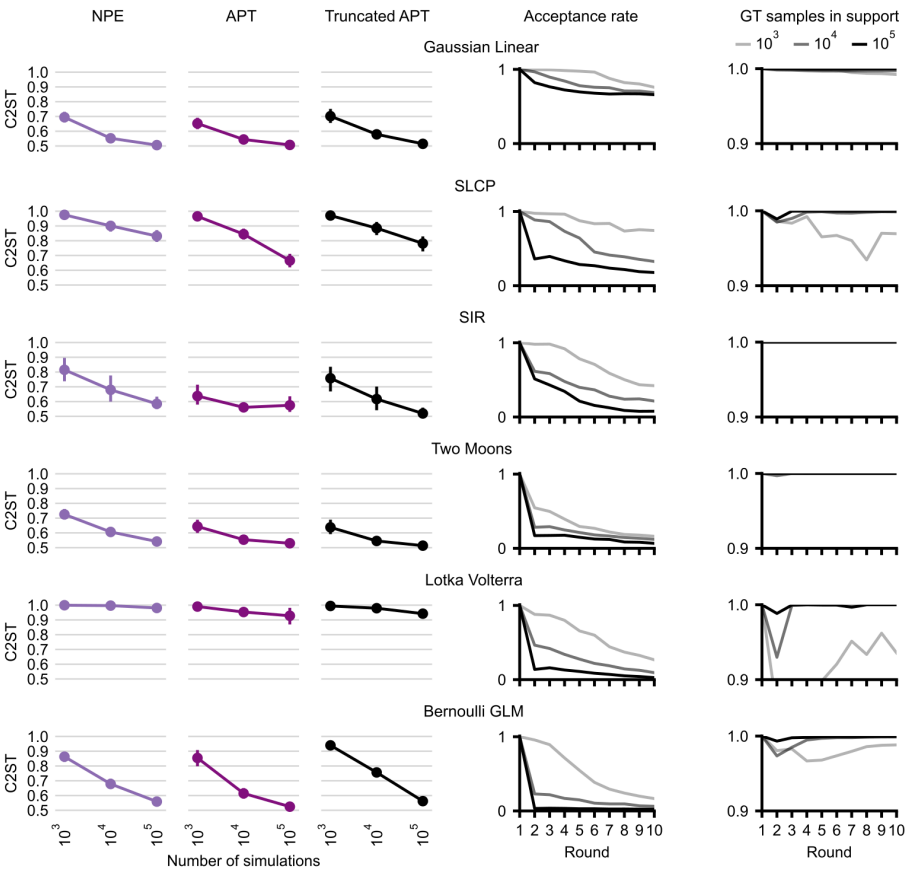


Figure 14: **Comparison between APT and truncated APT.** In order to investigate the effect of the truncated proposals on inference quality, we evaluated two versions of APT: In one scenario (middle column), we draw proposal samples from the previously estimated posterior, whereas in the second scenario (right column), we draw proposal samples from the truncated prior (with $\epsilon = 10^{-4}$). In both versions, we used the atomic loss function proposed in Greenberg et al. [2019]. Therefore, all differences stem from the different proposals. We also compare to standard NPE (trained with maximum-likelihood loss, left column). The two columns on the right are the same as in Fig. 4, evaluated for truncated APT.

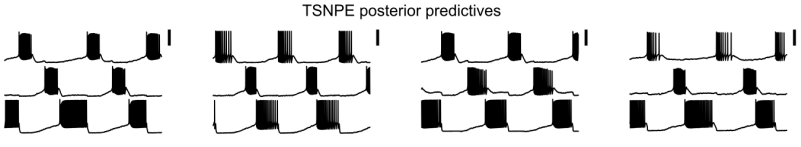


Figure 15: **Four posterior predictives of TSNPE applied to the pyloric network model.** The generated activity closely matches the summary statistics of the experimental data (Fig. 5a).

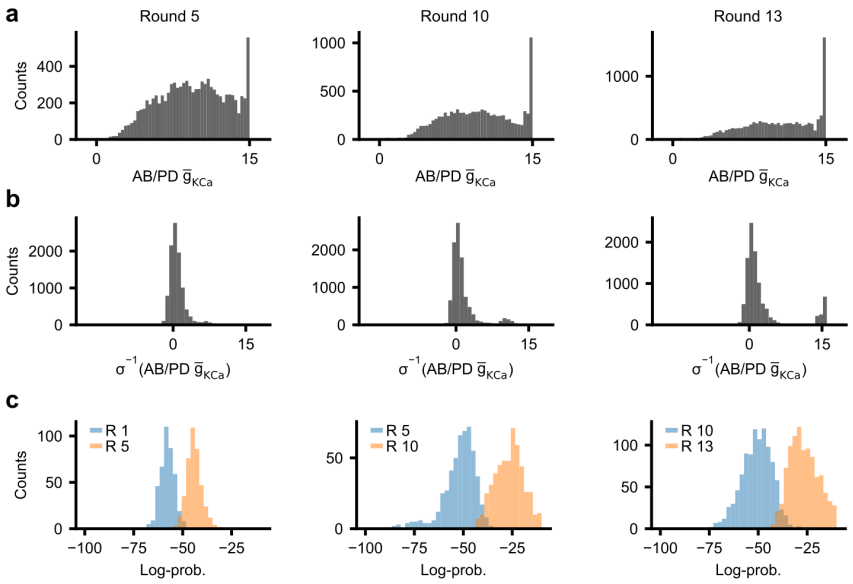


Figure 16: **Further explanation of issues when the parameter space is constrained.** (a) Marginal distribution of $\bar{g}_{\text{KCa}}$ of the AB/PD neuron when running APT for 5, 10, and 13 rounds. (b) Marginal distribution of this parameter, but when transformed into unconstrained space. This is the training data that the normalizing flow is 'effectively' seeing. (c) We evaluated those samples that were at the very right bounds of the marginal distribution under the current posterior (blue) as well as under the posterior from approximately 5 rounds before (orange). The samples have consistently lower log-probability under the previous posterior, which shows that the location of the leaking mass is moving.

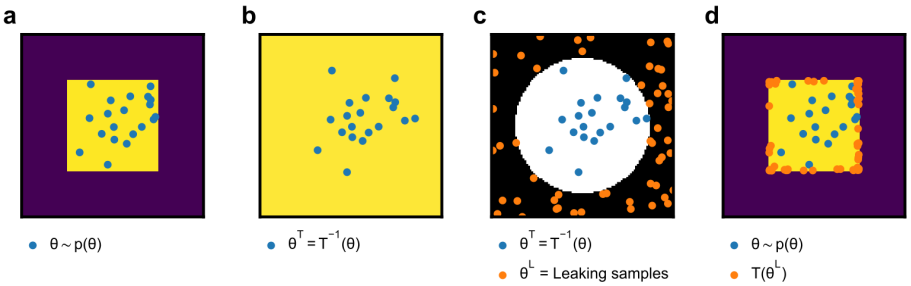


Figure 17: **Illustration of 'leakage' when prior is transformed to unconstrained space.** Note that these results are not based on an actual run of APT, but are purely for illustrative purposes. (a) The prior is a two-dimensional distribution on a constrained space (yellow region). Samples from the prior are in blue. (b) When the prior is transformed with an inverse sigmoid, its support becomes unconstrained. (c) APT will fit the posterior within the region of the training data (white region, blue samples). Outside of the training data, the trained density estimator can put arbitrary mass without affecting the loss (black region). Samples from this region are in orange. (d) When these 'leaking' samples (obtained with APT) are transformed back into constrained space, the 'leaking' mass ends up on the bounds of the prior.

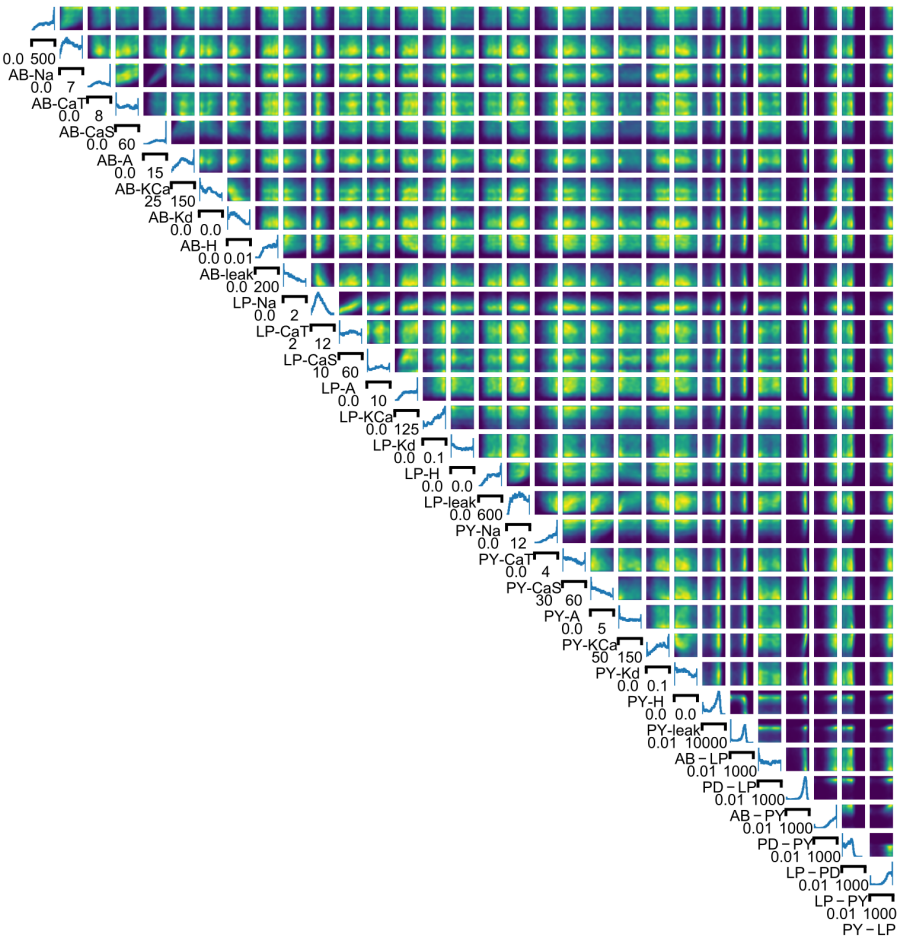


Figure 18: **Posterior distribution inferred by APT for the pyloric network model.**

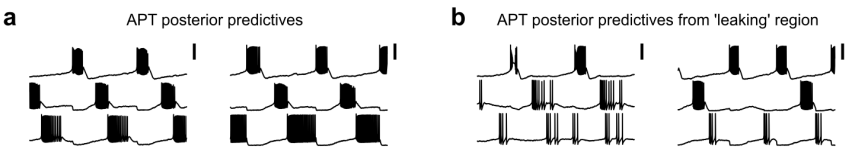


Figure 19: **Failure of APT when transformed to constrained space.** (a) Activity generated by two parameter sets sampled from the posterior distribution obtained with APT. We ensured that the shown parameter sets are not sampled from the very bounds of posterior distribution (i.e., that they were not sampled from the peak shown in Fig. 5c). (b) Activity generated by two parameter sets sampled from the posterior distribution obtained with APT. We specifically selected parameter sets whose value of $\overline{g}_{\mathrm{KCa}}$ in the AB-PD neuron was above 14.999 (i.e., those samples which are in the peak shown in Fig. 5c).

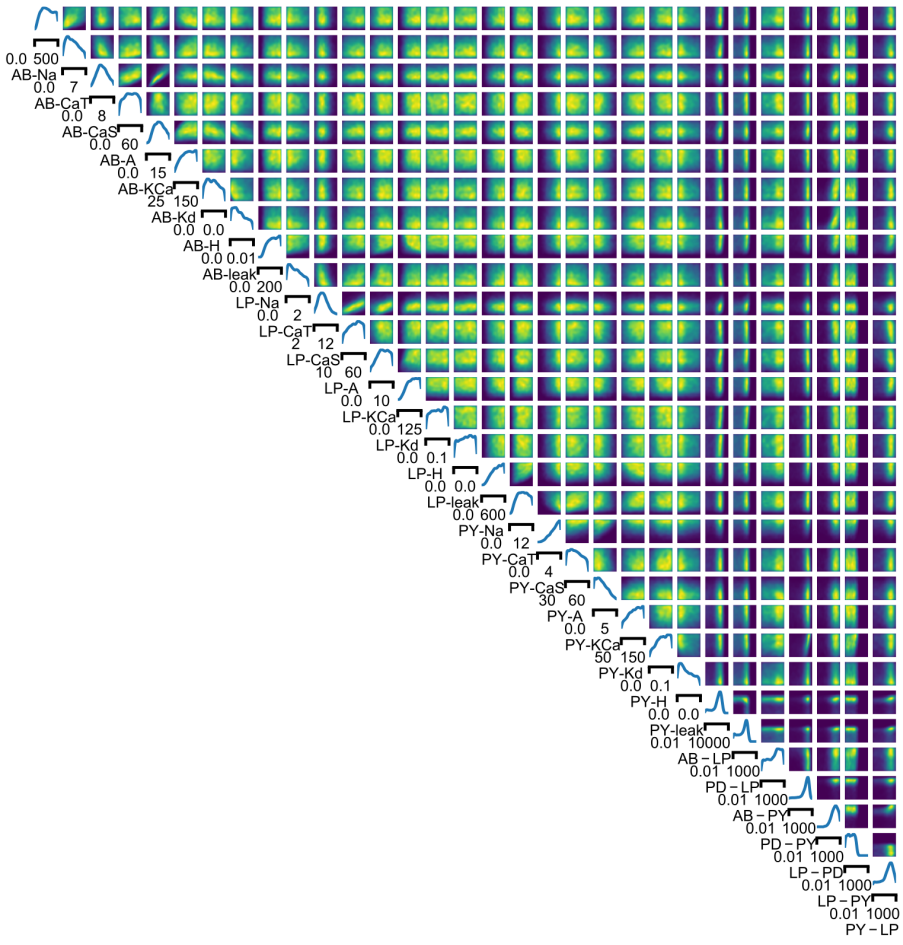


Figure 20: **Posterior distribution inferred by TSNPE for the pyloric network model.**

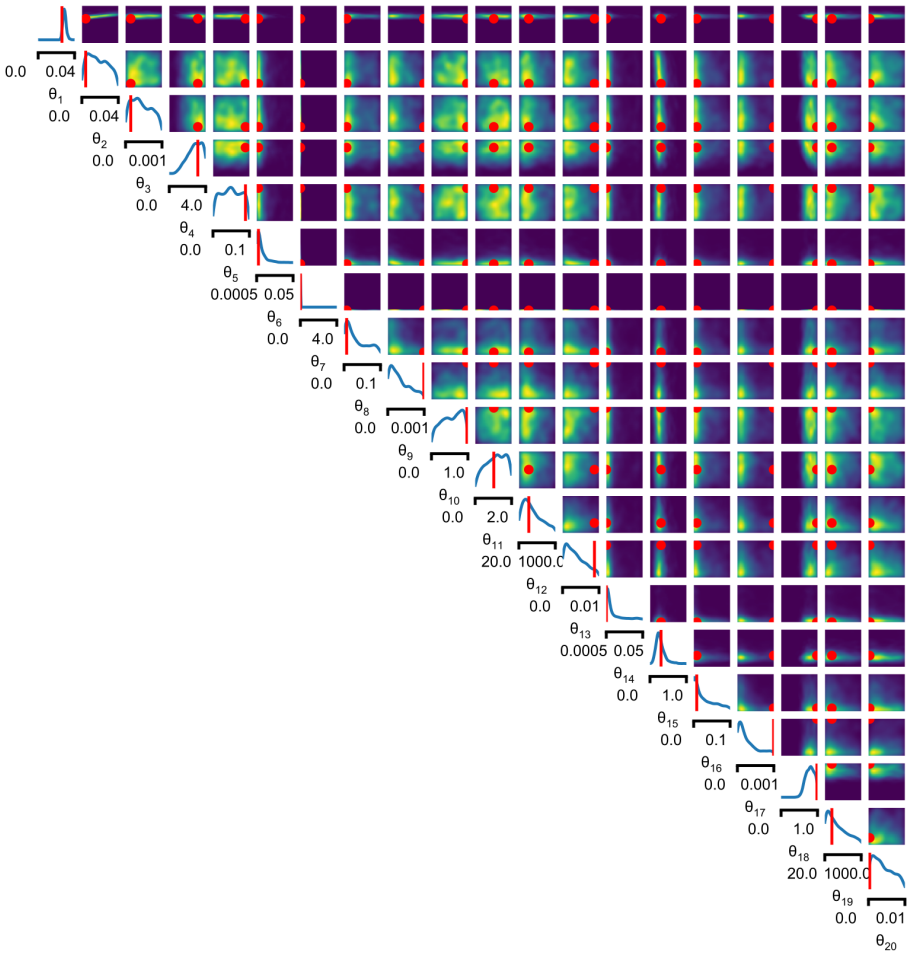


Figure 21: **Full posterior distribution over biophysical parameters for the layer 5 pyramidal cell.** Parameter names can be found in Appendix Sec. 6.14. Red dots are the ground-truth parameters.

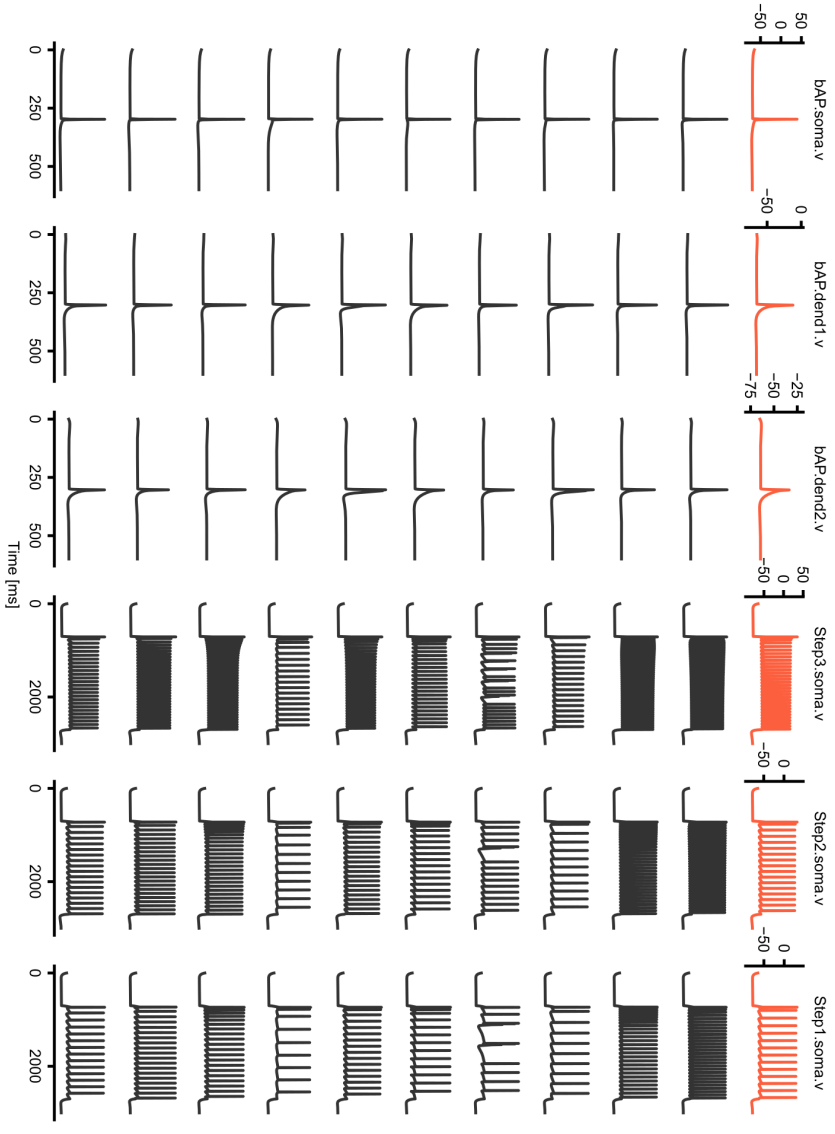


Figure 22: **Observed data (orange, top row) and ten posterior predictives obtained with TSNPE (black).**

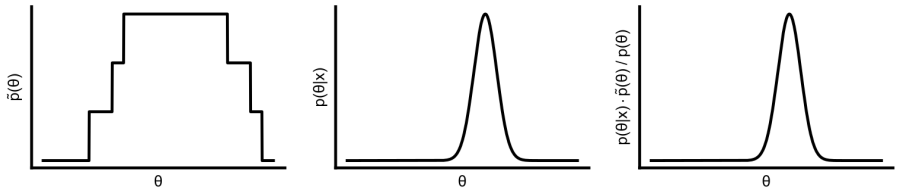


Figure 23: **Truncated proposals when pooling data from multiple rounds.** Assume a uniform prior. Left: Proposal that is the average of the proposal over three rounds. Middle: true posterior. Right: the approximate posterior converges to $p(\boldsymbol{\theta}|\mathbf{x}_o)\frac{\tilde{p}(\boldsymbol{\theta})}{p(\boldsymbol{\theta})}$ [Papamakarios and Murray, 2016, Lueckmann et al., 2017, Greenberg et al., 2019]. This matches the true posterior even though the proposal is not constant *outside* of the HPR_ϵ of the true posterior.

Generalized Bayesian Inference for Scientific Simulators via Amortized Cost Estimation

Richard Gao[*1,2]
r.dg.gao@gmail.com

Michael Deistler[*1,2]
michael.deistler@uni-tuebingen.de

Jakob H. Macke[1,2,3]
jakob.macke@uni-tuebingen.de

[1]Machine Learning in Science, Excellence Cluster Machine Learning, University of Tübingen
[2]Tübingen AI Center
[3]Department Empirical Inference, Max Planck Institute for Intelligent Systems
Tübingen, Germany
[*]Equal contributions.

Abstract

Simulation-based inference (SBI) enables amortized Bayesian inference for simulators with implicit likelihoods. But when we are primarily interested in the quality of predictive simulations, or when the model cannot exactly reproduce the observed data (i.e., is misspecified), targeting the Bayesian posterior may be overly restrictive. Generalized Bayesian Inference (GBI) aims to robustify inference for (misspecified) simulator models, replacing the likelihood-function with a cost function that evaluates the goodness of parameters relative to data. However, GBI methods generally require running multiple simulations to estimate the cost function at each parameter value during inference, making the approach computationally infeasible for even moderately complex simulators. Here, we propose amortized cost estimation (ACE) for GBI to address this challenge: We train a neural network to approximate the cost function, which we define as the expected distance between simulations produced by a parameter and observed data. The trained network can then be used with MCMC to infer GBI posteriors for any observation without running additional simulations. We show that, on several benchmark tasks, ACE accurately predicts cost and provides predictive simulations that are closer to synthetic observations than other SBI methods, especially for misspecified simulators. Finally, we apply ACE to infer parameters of the Hodgkin-Huxley model given real intracellular recordings from the Allen Cell Types Database. ACE identifies better data-matching parameters while being an order of magnitude more simulation-efficient than a standard SBI method. In summary, ACE combines the strengths of SBI methods and GBI to perform robust and simulation-amortized inference for scientific simulators.

1 Introduction

Mechanistic models expressed as computer simulators are used in a wide range of scientific domains, from astronomy, geophysics, to neurobiology. The parameters of the simulator, $\boldsymbol{\theta}$, encode mechanisms of interest, and simulating different parameter values produces different outputs, i.e., $\text{sim}(\boldsymbol{\theta}_i) \rightarrow \mathbf{x}_i$, where each model-simulation $\mathbf{x}_i$ can be compared to experimentally observed data, $\mathbf{x}_o$. Using such simulators, we can quantitatively reason about the contribution of mechanisms behind experimental

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

measurements. But to do so, a key objective is often to find all those parameter values that can produce simulations consistent with observed data.

One fruitful approach towards this goal is simulation-based inference (SBI) [1], which makes it possible to perform Bayesian inference on such models by interpreting simulator outputs as samples from an implicit likelihood [2], $\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})$. Standard Bayesian inference targets the parameter posterior distribution given observed data, i.e., $p(\boldsymbol{\theta}|\mathbf{x}_o) = \frac{p(\mathbf{x}_o|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{x}_o)}$, where $p(\boldsymbol{\theta})$ captures prior knowledge and constraints over model parameters, and the likelihood function $p(\mathbf{x}_o|\boldsymbol{\theta})$ is evaluated as a function of $\boldsymbol{\theta}$ for a fixed $\mathbf{x}_o$. SBI methods can differ in whether they aim to approximate the likelihood [3, 4, 5, 6] or the posterior directly [7, 8, 9], and can be *amortized*, i.e., do not require new simulations and retraining for new data [10, 11]. In the end, each method provides samples from the posterior, which are all, in theory, capable of producing simulations that are *identical* to the observation we condition on. Furthermore, by definition, the posterior probability of drawing a sample scales as the product of its prior probability and, critically, the likelihood that this sample can produce a simulation that is *exactly equal* to the observation.

However, targeting the exact posterior may be overly restrictive. In many inference scenarios, modelers are primarily interested in obtaining a diverse collection of parameter values that can explain the observed data. This desire is also reflected in the common usage of posterior predictive checks, where seeing predictive simulations that resemble the data closely (in some specific aspects) is used to gauge the success of the inference process. In particular, it is often clear that the scientific model is only a coarse approximation to the data-generating process, and in some cases even cannot generate data-matching simulations, i.e., is misspecified [12]. For example, in the life-sciences, it is not uncommon to use idealized, theoretically motivated models with few parameters, and it would be unrealistic to expect that they *precisely* capture observations of highly complex biological systems. In such cases, or in cases where the model is fully deterministic, it is nonsensical to use the probability of exactly reproducing the data. In contrast, it would still be useful to find parameter values that produce simulations which are 'close enough', or as close as possible to the data. Therefore, instead of sampling parameters according to *how often* they produce simulations that match the data *exactly*, many use cases call for sampling parameters according to *how closely* their corresponding simulations reproduce the observed data.

Generalized Bayesian Inference (GBI) [13] offers a principled way to do so by replacing the (log) likelihood function with a cost function that simply scores a parameter given an observation, such as the expected distance between $\mathbf{x}_o$ and all possible simulations $\mathbf{x}$ produced by $\boldsymbol{\theta}_i$ (Fig. 1). Several recent works have leveraged this framework to perform inference for models with implicit or intractable likelihoods, especially to tackle model misspecification: Matsubara et al. [14] use a Stein Discrepancy as the cost function (which requires the evaluation of an unnormalized likelihood and multiple i.i.d. data samples), and Cherief-Abdellatif et al. and Dellaporta et al. [15, 16] use simulator samples to estimate maximum mean discrepancy and directly optimize over this cost function via stochastic gradient descent (which requires a differentiable simulator). More broadly, cost functions such as scoring rule estimators have been used to generalize approximate Bayesian computation (ABC) [17] and synthetic likelihood approaches [3, 18], where the Monte Carlo estimate requires (multiple) simulations from $p(\mathbf{x}|\boldsymbol{\theta})$.

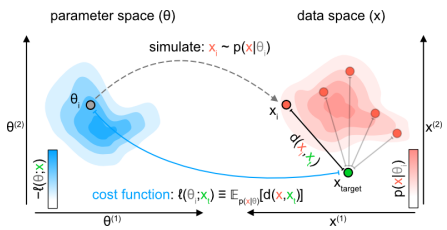


Figure 1: **Estimating cost from simulations.** Using the expected distance between simulated and target data as the cost function, GBI assigns high probability to parameter values that, on average, produce simulations that are close—but not necessarily equal—to the observation.

Thus, existing GBI approaches for SBI either require many simulations to be run during MCMC sampling of the posterior (similar to classical ABC methods), or are limited to differentiable simulators. Moreover, performing inference for new observations requires re-running simulations, rendering such methods simulation-inefficient and expensive at inference-time, and ultimately impractical for scientific simulators with even moderate computational burden.

We here propose to perform GBI for scientific simulators with **amortized cost estimation (ACE)**, which inherits the flexibility of GBI but amortizes the overhead of simulations by training a neural

network to predict the cost function for any parameter-observation pair. We first outline the GBI formalism in Sec. 2, then introduce ACE in Sec. 3. In Sec. 4, we show that ACE provides GBI posterior predictive simulations that are close to synthetic observations for a variety of benchmark tasks, especially when the simulator is misspecified. We showcase its real-world applicability in Sec. 5: using experimental data from the Allen Cell Types Database, ACE successfully infers parameters of the Hodgkin-Huxley single-neuron simulator with superior predictive performance and an order of magnitude higher simulation efficiency compared to neural posterior estimation [10]. Finally, we discuss benefits and limitations of GBI and ACE, and related work (Sec. 6).

2 Background

To construct the GBI posterior, the likelihood, $p(\mathbf{x}_o|\boldsymbol{\theta})$, is replaced by a 'generalized likelihood function', $L(\boldsymbol{\theta}; \mathbf{x}_o)$, which does not need to be a probabilistic model of the data-generating process, as long as it can be evaluated for any pair of $\boldsymbol{\theta}$ and $\mathbf{x}_o$. Following the convention in Bissiri et al. [13], we define $L(\boldsymbol{\theta}; \mathbf{x}_o) \equiv e^{-\beta \ell(\boldsymbol{\theta}; \mathbf{x}_o)}$, where $\ell(\boldsymbol{\theta}; \mathbf{x}_o)$ is a cost function that encodes the quality of $\boldsymbol{\theta}$ relative to an observation $\mathbf{x}_o$, and β is a scalar inverse temperature hyperparameter that controls how much the posterior weighs the cost relative to the prior. Thus, the GBI posterior can be written as

$$p(\boldsymbol{\theta}|\mathbf{x}_o) \propto e^{-\beta \ell(\boldsymbol{\theta}; \mathbf{x}_o)} p(\boldsymbol{\theta}). \tag{1}$$

As noted previously [13], if we define $\ell(\boldsymbol{\theta}; \mathbf{x}_o) \equiv -\log p(\mathbf{x}_o|\boldsymbol{\theta})$ (i.e., self-information) and $\beta = 1$, then we recover the standard Bayesian posterior, and "tempered" or "power" posterior for $\beta \neq 1$ [19, 20]. The advantage of GBI is that, instead of adhering strictly to the (implicit) likelihood, the user is allowed to choose arbitrary cost functions to rate the goodness of $\boldsymbol{\theta}$ relative to an observation $\mathbf{x}_o$, which is particularly useful when the simulator is misspecified. Previous works have referred to $\ell(\boldsymbol{\theta}; \mathbf{x}_o)$ as risk function [21], loss function [13], or (proper) scoring rules when they satisfy certain properties [22, 18] (further discussed in Section 6.1). Here we adopt 'cost' to avoid overloading the terms 'loss' and 'score' in the deep learning context.

3 Amortized Cost Estimation for GBI

3.1 Estimating cost function with neural networks

In this work, we consider cost functions that can be written as an expectation over the likelihood:

$$\ell(\boldsymbol{\theta}; \mathbf{x}_o) \equiv \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}, \mathbf{x}_o)] = \int_{\mathbf{x}} d(\mathbf{x}, \mathbf{x}_o) p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x}. \tag{2}$$

Many popular cost functions and scoring rules can be written in this form, including the average mean-squared error (MSE) [13], the maximum mean discrepancy (MMD^2) [23], and the energy score (ES) [22] (details in Appendix A2). While $\ell(\boldsymbol{\theta}; \mathbf{x}_o)$ can be estimated via Monte Carlo sampling, doing so in an SBI setting is simulation-inefficient and time-intensive, since the inference procedure must be repeated for every observation $\mathbf{x}_o$, and simulations must be run in real-time during MCMC sampling of the posterior. Furthermore, this does not take advantage of the structure in parameter-space or data-space around neighboring points that have been simulated.

We propose to overcome these limitations by training a regression neural network (NN) to learn $\ell(\boldsymbol{\theta}; \mathbf{x}_o)$. Our first insight is that cost functions of the form of Eq. 2 can be estimated from a dataset consisting of pairs of parameters and outputs—in particular, from *a single* simulation run per $\boldsymbol{\theta}$ for MSE, and finitely many simulation runs for MMD^2 and ES. We leverage the well-known property that NN regression converges to the conditional expectation of the data labels given the data: If we compute the distances $d(\mathbf{x}, \mathbf{x}_o)$ between a single observation $\mathbf{x}_o$ and every $\mathbf{x}$ in our dataset, then a neural network $f_\phi(\cdot)$ trained to predict the distances given parameters $\boldsymbol{\theta}$ will denoise the noisy distance labels $d(\mathbf{x}, \mathbf{x}_o)$ and converge onto the desired cost $f_\phi(\boldsymbol{\theta}) \to \ell(\boldsymbol{\theta}; \mathbf{x}_o) = \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}, \mathbf{x}_o)]$, approximating the cost of any $\boldsymbol{\theta}$ relative to $\mathbf{x}_o$ (see Appendix A3.1 for formal statement and proof).

3.2 Amortizing over observations

As outlined above, a regression NN will converge onto the cost function $\ell(\boldsymbol{\theta}; \mathbf{x}_o)$ for a particular observation $\mathbf{x}_o$. However, naively applying this procedure would require retraining of the network

for any new observation $\mathbf{x}_o$, which prevents application of this method in time-critical or high-throughput scenarios. We therefore propose to *amortize* cost estimation over a target distribution $p(\mathbf{x}_t)$. Specifically, a NN which receives as input a parameter $\boldsymbol{\theta}$ and an independently sampled target datapoint $\mathbf{x}_t$ will converge to $\ell(\boldsymbol{\theta}; \mathbf{x}_t)$ for all $\mathbf{x}_t$ on the support of the target distribution (Fig. 2a,b), enabling estimation of the cost function for any pair of $(\boldsymbol{\theta}, \mathbf{x}_t)$ (see Appendix A3.2 for formal statement and proof).

Naturally, we use the already simulated $\mathbf{x} \sim p(\mathbf{x})$ as target data during training, and therefore do not require further simulations. In order to have good accuracy on potentially misspecified observations, however, such datapoints should be within the support of the target distribution. Thus, in practice, we augment this target dataset with noisy simulations to broaden the support of $p(\mathbf{x}_t)$. Furthermore, if the set of observations (i.e., real data) is known upfront, they can also be appended to the target dataset during training. Lastly, to keep training efficient and avoid quadratic scaling in the number of simulations, we randomly subsample a small number of $\mathbf{x}_t$ per $\boldsymbol{\theta}$ in each training epoch (2 in our experiments), thus ensuring linear scaling as a function of simulation budget. Fig. 2a,b summarizes dataset construction and network training for ACE (details in Appendix A4.1).

3.3 Sampling from the generalized posterior

Given a trained cost estimation network $f_\phi(\cdot, \cdot)$, an observed datapoint $\mathbf{x}_o$, and a user-selected inverse temperature β, the generalized posterior probability (Eq. 1) can be computed up to proportionality for any $\boldsymbol{\theta}$: $p(\boldsymbol{\theta}|\mathbf{x}_o) \propto \exp(-\beta \cdot f_\phi(\boldsymbol{\theta}, \mathbf{x}_o))p(\boldsymbol{\theta})$ and, thus, this term can be sampled with MCMC (Fig. 2c). The entire algorithm is summarized in Algorithm 1.

Algorithm 1: Generalized Bayesian Inference with Amortized Cost Estimation (ACE)

Inputs: prior $p(\boldsymbol{\theta})$, simulator with implicit likelihood $p(\mathbf{x}|\boldsymbol{\theta})$, number of simulations N, feedforward NN f_ϕ with parameters ϕ, NN learning rate η, distance function $d(\cdot, \cdot)$, noise level σ, number of noise-augmented samples S, inverse temperature β, number of target datapoints per $\boldsymbol{\theta}$ N_{target}, K observations $\mathbf{x}_o^{(1,\dots,K)}$.
Outputs: M samples from generalized posteriors given K observations.

Generate dataset:
sample prior and simulate: $\boldsymbol{\theta}, \mathbf{x} \leftarrow \{\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}), \mathbf{x}_i \sim p(\mathbf{x}|\boldsymbol{\theta}_i)\}_{i:1\dots N}$
add noise and concatenate: $\mathbf{x}_{\text{target}} = [\mathbf{x},\ \mathbf{x}_{1\dots S} + \boldsymbol{\epsilon},\ \mathbf{x}_o], \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$

Training:
while *not converged* **do**
  for $(\boldsymbol{\theta}, \mathbf{x})$ *in batch* **do**
    $\mathbf{x}_t^{\text{used}} \leftarrow$ sample N_{target} datapoints from $\mathbf{x}_{\text{target}}$
    for $\mathbf{x}_t$ *in* $\mathbf{x}_t^{used}$ **do**
      $\mathcal{L} \leftarrow \mathcal{L} + (f_\phi(\boldsymbol{\theta}, \mathbf{x}_t) - d(\mathbf{x}, \mathbf{x}_t))^2$
  $\phi \leftarrow \phi - \eta \cdot \text{Adam}(\nabla_\phi \mathcal{L})$; `// and reset L to zero`

Sampling:
for $k \in [1, ..., K]$ **do**
  Draw M samples, with MCMC, from: $\exp(-\beta \cdot f_\phi(\boldsymbol{\theta}, \mathbf{x}_o^{(k)}))\ p(\boldsymbol{\theta})$

3.4 Considerations for choosing the value of β

We note that the choice of value for β is an important decision, though this is an issue not only for ACE but GBI methods in general [13]. A good "baseline" value for β is such that the average distance across a subset of the training data is scaled to be in the same range as the (log) prior probability, both of which can be computed on prior simulations. From there, increasing β sacrifices sample diversity for predictive distance, and as β approaches infinity, posterior samples converge onto the minimizer of the cost function. In practice, we recommend experimenting with a range of (roughly) log-spaced values since, as we show below, predictive sample quality tend to improve with increasing β.

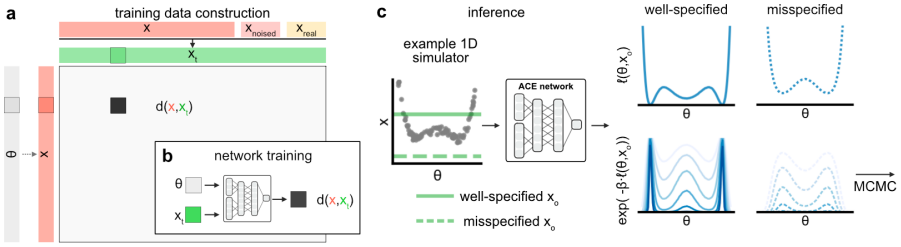


Figure 2: **Schematic of dataset construction, network training, and inference. (a-b)** The neural network is trained to predict the distance between pairs of $\mathbf{x}$ (red) and $\mathbf{x}_t$ (green), as a noisy sample of the cost function (i.e., expected distance) evaluated on $\boldsymbol{\theta}$ (grey) and $\mathbf{x}_t$. **(c)** At inference time, the trained ACE network predicts the cost for any parameter $\boldsymbol{\theta}$ given observation $\mathbf{x}_o$ (top row), which is used to evaluate the GBI posterior under different β (bottom row, darker for larger β) for MCMC sampling without running additional simulations. The distance is well-defined and can be approximated even when the simulator is misspecified (dashed lines).

4 Benchmark experiments

4.1 Experiment setup

Tasks We first evaluated ACE on four benchmark tasks (modified from Lueckmann et al. [24]) with a variety of parameter- and data-dimensionality, as well as choice of distance measure: (1) **Uniform 1D**: 1D $\boldsymbol{\theta}$ and $\mathbf{x}$, the simulator implements an even polynomial with uniform noise likelihood, uniform prior (Fig. 2c); (2) **2 Moons**: 2D $\boldsymbol{\theta}$ and $\mathbf{x}$, simulator produces a half-circle with constant mean radius and radially uniform noise of constant width, translated as a function of $\boldsymbol{\theta}$, uniform prior; (3) **Linear Gaussian**: 10D $\boldsymbol{\theta}$ and $\mathbf{x}$, Gaussian model with mean $\boldsymbol{\theta}$ and fixed covariance, Gaussian prior; (4) **Gaussian Mixture**: 2D $\boldsymbol{\theta}$ and $\mathbf{x}$, simulator returns five i.i.d. samples from a mixture of two Gaussians, both with mean $\boldsymbol{\theta}$, and fixed covariances, one with broader covariance than the other, uniform prior.

For the first three tasks, we use the mean-squared error between simulation and observation as the distance function. For the Gaussian Mixture task, we use maximum mean discrepancy (MMD^2) to measure the statistical distance between two sets of five i.i.d. samples. Importantly, for each of the four tasks, we can compute the integral in Eq. 2 either analytically or accurately capture it with quadrature over $\mathbf{x}$. Hence, we obtain the true cost $\ell(\boldsymbol{\theta}; \mathbf{x}_o)$ and subsequently the 'ground-truth' GBI posterior (with Eq. 1), and use that to draw, for each value of β and $\mathbf{x}_o$, 5000 samples (GT-GBI, black in Fig. 3). See Appendix A4.2 for more detailed descriptions of tasks and distance functions.

Training data For each task, we simulate 10,000 pairs of $(\boldsymbol{\theta}, \mathbf{x})$ and construct the target dataset as in Fig. 2a, with 100 additional noise-augmented targets and 20 synthetic observations—10 well-specified and 10 misspecified—for a total of 10120 $\mathbf{x}_t$ data points. Well-specified observations are additional prior predictive samples, while misspecified observations are created by moving prior predictive samples outside the boundaries defined by the minimum and maximum of 100,000 prior predictive simulations (e.g., by successively adding Gaussian noise). See Appendix A4.3 for details.

Test data To evaluate inference performance, we use ACE to sample approximate GBI posteriors conditioned on 40 different synthetic observations, 20 of which were included in the target dataset $\mathbf{x}_t$, and 10 additional well-specified and misspecified observations which were not included in the target dataset. We emphasize that including observations in the target data is not a case of test data leakage, but represents a real use case where some experimental data which one wants to perform inference on are already available, while the network should also be amortized for unseen observations measured after training. Nevertheless, we report in Fig. 3 results for 'unseen' observations, i.e., not in the target dataset. Results are almost identical for those that were in the target dataset (Appendix A1). We drew 5000 posterior samples per observation, for 3 different β values for each task.

Metrics We are primarily interested in two aspects of performance: approximate posterior predictive distance and cost estimation accuracy. First, as motivated above, we want to find parameter

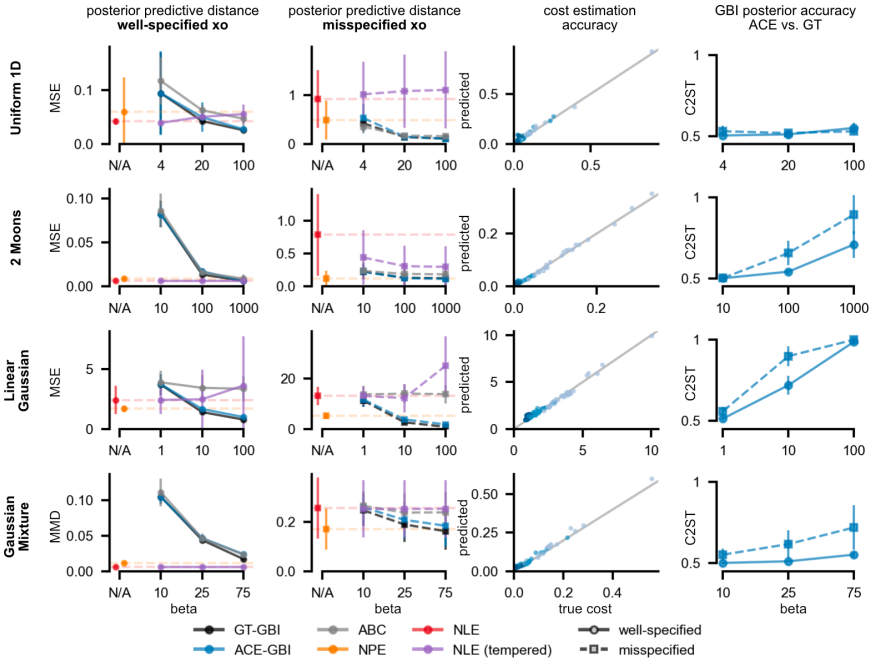


Figure 3: **Performance on benchmark tasks.** ACE obtains posterior samples with low average distance to observations, and accurately estimates cost function. **Rows:** results for each task. **Columns:** average predictive distance compared to SBI methods and GT (1st and 2nd), cost estimation accuracy evaluated on ACE posterior samples for different β (lighter blue shades are lower values of β) (3rd), and C2ST accuracy relative to GT GBI posterior (4th, lower is better).

configurations which produce simulations that are as close as possible to the observation, as measured by the task-specific distance function. Therefore, we simulate using each of the 5000 ACE GBI posterior samples, and compute the average distance between predictive simulations and the observation. Mean and standard deviation are shown for well-specified and misspecified observations separately below (Fig. 3, 1st and 2nd columns). Second, we want to confirm that ACE accurately approximates $\ell(\boldsymbol{\theta}; \mathbf{x}_o)$, which is a prerequisite for correctly inferring the GBI posterior. Therefore, we compare the ACE-predicted and true cost across 5000 samples from each GBI posterior, as well as the classifier 2-sample test (C2ST, [25, 24]) score between the ACE approximate and ground-truth GBI posterior (Fig. 3, 3rd and 4th columns). Note that cost estimation accuracy can be evaluated for parameter values sampled in any way (e.g., from the prior), but here we evaluate accuracy as samples become more concentrated around good parameter values, i.e., from GBI posteriors with increasing β. We expect that these tasks become increasingly challenging with higher values of β, since these settings require the cost estimation network to be highly accurate in tiny regions of parameter-space.

Other algorithms As a comparison against SBI methods that target the standard Bayesian posterior (but which nevertheless might produce good predictive samples), we also tested approximate Bayesian computation (ABC), neural posterior estimation (NPE, [7]), and neural likelihood estimation (NLE, [4, 26]) on the same tasks. NPE and NLE were trained on the same 10,000 simulations, and 5000 approximate posterior samples were obtained for each $\mathbf{x}_o$. We used the amortized (single-round) variants of both as a fair comparison against ACE. Additionally, to rule out the possibility that ACE is simply benefiting from the additional inverse temperature hyperparameter, we implemented a "tempered" version of NLE by scaling the NLE-approximated log-likelihood term (by the same βs) during MCMC sampling. For ABC, we used the 10,000 training samples as a reference set, from which 50 were drawn as posterior samples with probability scaling inversely with the distance between their corresponding simulation and the observation, i.e., ABC with acceptance kernel [27].

4.2 Benchmark results

Overall, we see that for well-specified $\mathbf{x}_o$ (i.e., observations for which the simulator is well-specified), ACE obtains GBI posterior samples that achieve low average posterior predictive simulation distance across all four tasks, especially at high values of β (Fig. 3, 1st column). In comparison, ABC is worse for the Linear Gaussian task (which has a higher parameter dimensionality than all other tasks), whereas NPE, NLE, and tempered NLE achieve similarly low posterior predictive distances.

On misspecified observations, across all tasks and simulation-budgets (with the exception of Gaussian mixture on 10k simulations) we see that ACE achieves lower or equally low average posterior predictive simulation distance as neural SBI methods, even at moderate values of β (Fig. 3, 2nd column, Figs. A4, A5). This is in line with our intuition that ACE returns a valid and accurate cost even if the simulator is incapable of producing data anywhere near the observation, while Bayesian likelihood and posterior probabilities estimated by NLE and NPE are in these cases nonsensical [28, 29, 30, 31]. Furthermore, simply concentrating the approximate Bayesian posterior via tempering does not lead to more competitive performance than ACE on such misspecified observations, and is sometimes even detrimental (e.g., tempered NLE at high β on Uniform 1D and Linear Gaussian tasks). Therefore, we see that ACE can perform valid inference for a broad range of simulators, obtaining a distribution of posterior samples with predictive simulations close to observations, and is automatically robust against model-misspecification as a result of directly targeting the cost function.

For both well-specified and misspecified observations, ACE-GBI samples achieve posterior predictive distance very close to ground-truth (GT)-GBI samples, at all values of β (Fig. 3, 1st and 2nd column), suggesting that ACE is able to accurately predict the expected distance. Indeed, especially for low to moderate values of β, the ACE-predicted cost closely matches the true cost (Fig. 3, 3rd column, light blue for well-specified $\mathbf{x}_o$, Fig. A2 for misspecified). For higher values of β, ACE-predicted cost is still similar to true cost, although the error is, as expected, larger for very large β (Fig. 3, 3rd column, dark blue).

As a result, highly concentrated generalized posteriors are estimated with larger (relative) discrepancies, which is reflected in the classifier 2-sample score (C2ST) between ACE and GT GBI posteriors (Fig. 3, 4th column): ACE posterior samples are indistinguishable from GT samples at low β, even for the 10D Linear Gaussian task, but becomes less accurate with increasing β. Nevertheless, predictive simulation distance dramatically increases with β even when ACE is less accurate, suggesting that sampling to explicitly minimize a cost function which targets parameters with data-similar simulations is a productive goal. Relative performance results across algorithms are qualitatively similar when using a training simulation budget of 200 (Fig. A4) and 1000 (Fig. A5), but ABC required a sufficiently high simulation budget and performed poorly for 1000 training simulations or less.

5 Hodgkin-Huxley inference from Allen Cell Types Database recordings

Finally, we applied ACE to a commonly used scientific simulator and real data: we used a single-compartment Hodgkin-Huxley (HH) simulator from neuroscience and aimed to infer eight parameters of the simulator given electrophysiological recordings from the Allen Cell Types Database [32, 33, 34]. While this simulator can generate a broad range of voltage traces, it is still a crude approximation to *real* neurons: it models only a subset of ion channels, it ignores the spatial structure of neurons, and it ignores many intracellular mechanisms [35]. It has been demonstrated that parameters of the HH-model given *synthetic* recordings can be efficiently estimated with standard NPE [10], but estimating parameters given *experimental* recordings has been challenging [36] and has required ad-hoc changes to the inference procedure (e.g., Bernaerts et al. [37] added noise to the summary statistics, and Gonçalves et al. [10] used a custom multi-round scheme with a particular choice of density estimator). We will demonstrate that ACE can successfully perform simulation-amortized inference given experimental recordings from the Allen Cell Types Database (Fig. 4a).

We trained NPE and ACE given 100K prior-sampled simulations (details in Appendix A4.4). After training, ACE accurately predicts the true cost of parameters given experimental observations (Fig. 4b). We then used slice sampling to draw samples from the GBI posterior for three different values of $\beta = \{25, 50, 100\}$ and for ten observations from the Allen Cell Types Database. Interestingly, the marginal distributions between NPE and ACE posteriors are very similar, especially for rather low values of β (Fig. 4c, cornerplot in Appendix Fig. A7). The quality of posterior predictive samples, however, strongly differs between NPE and ACE: across the ten observations from the Allen

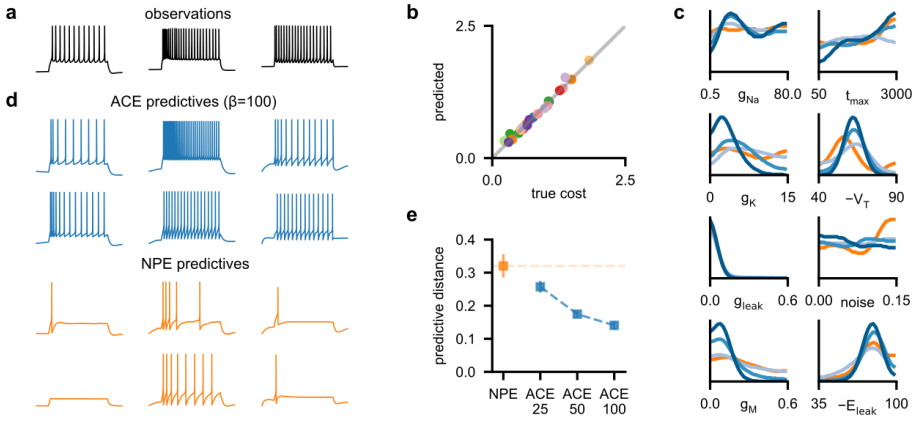


Figure 4: **Applicaton of ACE to Allen data.** **(a)** Three observations from the Allen Cell Types Database. **(b)** True cost (evaluated as Monte-Carlo average over 10 simulations) per $\boldsymbol{\theta}$ vs ACE-predicted cost. Colors are different observations. **(c)** Marginals of posterior distributions for NPE (orange) and ACE (shades of blue. Light blue: $\beta = 25$, medium blue: $\beta = 50$, dark blue: $\beta = 100$). **(d)** Top: Two GBI predictive samples for each observation. Bottom: Two NPE predictive samples. Additional samples in Appendix A8-A11. **(e)** Average predictive distance to observation for NPE and ACE with $\beta = \{25, 50, 100\}$.

Cell Types database, only 35.6% of NPE posterior predictives produced more than five spikes (all observations have at least 12 spikes), whereas the ACE posterior predictives closely match the data, even for low values of β (Fig. 4d, samples for all observations and all β values in Figs. A8,A9,A10 and for NPE in Fig. A11. 66% ($\beta = 25$), 87% ($\beta = 50$), 96% ($\beta = 100$) of samples have more than five spikes). Indeed, across all ten observations, the average posterior predictive distance of ACE was significantly smaller than that of NPE, and for large values of β the distance is even less than half (Fig. 4e). Finally, for rejection-ABC, only the top 35 samples (out of the full training budget of 100K simulations) had a distance that is less than the *average* posterior predictive distance achieved by ACE.

To investigate these differences between NPE and ACE, we also evaluated NPE posterior predictive performance on synthetic data (prior predictives) and found that it had an average predictive distance of 0.189, which roughly matches the performance of ACE on the experimental observations (0.174 for β=50). This suggest that, in line with previous results [37], NPE indeed struggles with *experimental* observations, for which the simulator is inevitably imperfect. We then trained NPE with 10 times more simulations (1M in total). With this increased simulation budget, NPE performed significantly better than with 100K simulations, but still produced poorer predictive samples than ACE trained with 100K simulations (for $\beta = \{50, 100\}$), although the marginals were similar between NPE (1M) and ACE (100K) (Fig. A6, samples for all observations in Appendix Fig. A12).

Overall, these results demonstrate that ACE can successfully be applied to real-world simulators on which vanilla NPE fails. On the Hodgkin-Huxley simulator, ACE generates samples with improved predictive accuracy despite an order of magnitude fewer simulations and despite the marginal distributions being similar to those of NPE.

6 Discussion

We presented ACE, a method to perform distance-aware inference for scientific simulators within the Generalized Bayesian Inference (GBI) framework. Contrary to 'standard' simulation-based inference (SBI), our method does not target the Bayesian posterior, but replaces the likelihood function with a cost function. For real-world simulators, doing so can provide practical advantages over standard Bayesian inference:

First, the likelihood function quantifies the probability that a parameter generates data which *exactly* matches the data. However, in cases where the model is a rather crude approximation to the real system being studied, scientists might well want to include parameters that can generate data that is sufficiently close (but not necessarily identical) in subsequent analyses. Our method makes this possible, and is advantageous over other GBI-based methods since it is amortized over observations and the inverse temperature β. Second, many simulators are formulated as noise-free models, and it can be hard to define appropriate stochastic extensions (e.g., [38]). In these cases, the likelihood function is ill-defined and, in practice, this setting would require 'standard' SBI methods, whose density estimators are generally built to model continuous distributions, to model discrete jumps in the posterior density. In contrast, our method can systematically and easily deal with noise-free simulators, and in such situations more closely resembles parameter-fitting algorithms. Lastly, standard Bayesian inference is challenging when the model is misspecified, and the performance of neural network-based SBI methods can suffer drastically in this scenario [30].

6.1 Related work

GBI for Approximate Bayesian Computation Several studies have proposed methods that perform GBI on simulators with either an implicit (i.e., simulation-based) likelihood or an unnormalized likelihood. Wilkinson et al. [39] argued that rejection-ABC performs exact inference for a modified model (namely, one that appends an additive uniform error) instead of approximate inference for the original model. Furthermore, ABC with arbitrary probabilistic acceptance kernels can also be interpreted as having different error models, and Schmon et al. [17] integrate this view to introduce generalized posteriors for ABC, allowing the user to replace the hard-threshold kernel (i.e., ϵ-ball of acceptance) with an arbitrary loss function that measures the discrepancy between $\mathbf{x}$ and $\mathbf{x}_o$ for MCMC-sampling of the approximate generalized posterior.

Other recent GBI methods require a differentiable simulator [16, 15] or build tractable cost functions that can be sampled with MCMC [14, 18], but this still requires running simulations *at inference time* (i.e., during MCMC) and does not amortize the cost of simulations and does not reuse already simulated datapoints.

Finally, Bayesian Optimization for Likelihood-free Inference (BOLFI, [40]) and error-guided LFI-MCMC [41] are not cast as generalized Bayesian inference approaches, but are related to ACE. Similarly as in ACE, they train models (for BOLFI, a Gaussian process and, for error-guided LFI-MCMC, a classifier) to estimate the discrepancy between observation and simulation. In BOLFI, the estimator is then used to iteratively select new locations at which to simulate. However, contrary to ACE, neither of these two methods amortizes the cost of simulations over observations.

Misspecification-aware SBI Several other methods have been proposed to overcome the problem of misspecification in SBI: For example, Bernaerts et al. [37] add noise to the summary statistics in the training data, Ward et al. [42] use MCMC to make the misspecified data well-specified, and Kelly et al. [43] introduce auxiliary variables to shift the (misspecified) observation towards being well-specified. All of these methods, however, maintain that the inference result should be an 'as close as possible' version of the posterior distribution. Contrary to that, our method does *not* aim to obtain the Bayesian posterior distribution (which, for misspecified models, can often be nonsensical or even undefined if the evidence is zero), but is specifically targeted towards parameter regions that are a specified distance from the observation. More broadly, recent advances in uncertainty quantification in deep neural networks, where standard mean-predicting regression networks are supplemented with a uncertainty- or variance-predicting network [44, 45], may serve to further connect loss-minimizing deep learning with (misspecification-aware) SBI.

6.2 Limitations

While our method amortizes the cost of simulations and of training, it still requires another method to sample from the posterior distribution. We used multi-chain slice-sampling [46] for efficiency, but any other MCMC algorithm, as well as variational inference, could also be employed [47, 48]. While sampling incurs an additional cost, this cost is generally small in comparison to potentially expensive simulations.

In addition, our method can perform inference for distance functions which can be written as expectations over the likelihood. As we demonstrated, this applies to many popular and widely used distances. Our method can, however, not be applied to arbitrary distance functions (e.g., the minimum distance between all simulator samples and the observation). While the distances we investigated here are certainly useful to practioners, they do not necessarily fulfill the criterion of being 'proper' scoring rules [22, 18]. Furthermore, we note that the cost functions considered here by default give rise to unnormalized generalized likelihoods. Therefore, depending on whether the user aims to approximate the generalized posterior given the normalized or unnormalized likelihood, different MCMC schemes should be used in conjunction with ACE (standard MCMC vs. doubly-intractable MCMC, e.g., the Exchange algorithm [49]).

Compared to 'standard' SBI, GBI introduces an additional hyperparameter to the inference procedure, the inverse temperature β. This hyperparameter has to be set by the user and its choice strongly affects inference behaviour: low values of β will include regions of parameter-space whose data do not necessarily match the observation closely, whereas high values of β constrain the parameters to only the best-fitting parameter values. While we provide heuristics for selecting β, we acknowledge the inconvenience of an additional hyperparameter. However, our method is amortized over β, which makes exploration of different β values possible, and which could simplify automated methods for setting β, similar to works where β is taken as the exponent of the likelihood function [50].

Finally, as with any method leveraging deep neural networks, including neural density estimator-based SBI methods (such as NPE), sensitivity to the number of training samples and the dimensionality of the task should always be considered. As we demonstrate above, increasing simulation budget improves the performance of any algorithm, and a reasonable number of training simulations yielded improved performance on a real-world neuroscience application, while the amortization property shifts the cost of simulations up front. In addition, we consider tasks up to 10 dimensions here, as most existing SBI methods have been benchmarked and shown to perform adequately on such tasks [24], though it remains to be seen how ACE can extend to higher dimensional parameter-space and data-space and whether embedding networks will be similarly helpful.

7 Conclusion

We presented a method that performs generalized Bayesian inference with amortized cost estimation. Our method produces good predictive samples on several benchmark tasks, especially in the case of misspecified observations, and we showed that it allows amortized parameter estimation of Hodgkin-Huxley models given experimental recordings from the Allen Cell Types Database.

8 Acknowledgements

RG is supported by the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 101030918 (AutoMIND). MD is supported by the International Max Planck Research School for Intelligent Systems (IMPRS-IS). The authors are funded by the Machine Learning Cluster of Excellence, EXC number 2064/1–390727645. This work was supported by the Tübingen AI Center (Agile Research Funds), the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039A, and the German Research Foundation (DFG): SFB 1233, Robust Vision: Inference Principles and Neural Mechanisms, project number: 276693517.

We would like to thank Jan Boelts, Janne Lappalainen, and Auguste Schulz for feedback on the manuscript, Julius Vetter for feedback and discussion on proper scoring rules, Poornima Ramesh and Mackelab members for extensive discussions throughout the project, as well as Francois-Xavier Briol for suggestions on sampling doubly-intractable posteriors, and the reviewers for their constructive comments on the readability of the manuscript and suggestions for additional analyses.

References

[1] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020.

[2] Peter J Diggle and Richard J Gratton. Monte carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society. Series B, Statistical methodology*, 46(2):193–227, 1984.

[3] Simon N Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310):1102–1104, 2010.

[4] George Papamakarios, David Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 837–848. PMLR, 2019.

[5] Joeri Hermans, Volodimir Begy, and Gilles Louppe. Likelihood-free mcmc with amortized approximate ratio estimators. In *International Conference on Machine Learning*, pages 4239–4248. PMLR, 2020.

[6] Owen Thomas, Ritabrata Dutta, Jukka Corander, Samuel Kaski, and Michael U Gutmann. Likelihood-free inference by ratio estimation. *Bayesian Analysis*, 17(1):1–31, 2022.

[7] George Papamakarios and Iain Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. *Advances in neural information processing systems*, 29, 2016.

[8] David Greenberg, Marcel Nonnenmacher, and Jakob Macke. Automatic posterior transformation for likelihood-free inference. In *International Conference on Machine Learning*, pages 2404–2414. PMLR, 2019.

[9] Katalin Csilléry, Michael GB Blum, Oscar E Gaggiotti, and Olivier François. Approximate bayesian computation (abc) in practice. *Trends in ecology & evolution*, 25(7):410–418, 2010.

[10] Pedro J Gonçalves, Jan-Matthis Lueckmann, Michael Deistler, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F Podlaski, Sara A Haddad, Tim P Vogels, et al. Training deep neural density estimators to identify mechanistic models of neural dynamics. *Elife*, 9:e56261, 2020.

[11] Stefan T Radev, Ulf K Mertens, Andreas Voss, Lynton Ardizzone, and Ullrich Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[12] Stephen G Walker. Bayesian inference with misspecified models. *Journal of statistical planning and inference*, 143(10):1621–1633, October 2013.

[13] Pier Giovanni Bissiri, Chris C Holmes, and Stephen G Walker. A general framework for updating belief distributions. *Journal of the royal statistical society. series b, statistical methodology*, 78(5):1103, 2016.

[14] Takuo Matsubara, Jeremias Knoblauch, François-Xavier Briol, Chris Oates, et al. Robust generalised bayesian inference for intractable likelihoods. *arXiv preprint arXiv:2104.07359*, 2021.

[15] Badr-Eddine Cherief-Abdellatif and Pierre Alquier. MMD-Bayes: Robust bayesian estimation via maximum mean discrepancy. In Cheng Zhang, Francisco Ruiz, Thang Bui, Adji Bousso Dieng, and Dawen Liang, editors, *Proceedings of The 2nd Symposium on Advances in Approximate Bayesian Inference*, volume 118 of *Proceedings of Machine Learning Research*, pages 1–21. PMLR, December 2020.

[16] Charita Dellaporta, Jeremias Knoblauch, Theodoros Damoulas, and François-Xavier Briol. Robust bayesian inference for simulator-based models via the mmd posterior bootstrap. In *International Conference on Artificial Intelligence and Statistics*, pages 943–970. PMLR, 2022.

[17] Sebastian M Schmon, Patrick W Cannon, and Jeremias Knoblauch. Generalized posteriors in approximate bayesian computation. *arXiv preprint arXiv:2011.08644*, 2020.

[18] Lorenzo Pacchiardi and Ritabrata Dutta. Generalized bayesian likelihood-free inference using scoring rules estimators. *arXiv preprint arXiv:2104.03889*, 2021.

[19] Enzo Marinari and Giorgio Parisi. Simulated tempering: A new monte carlo scheme. May 1992.

[20] N Friel and A N Pettitt. Marginal likelihood estimation via power posteriors. *Journal of the Royal Statistical Society. Series B, Statistical methodology*, 70(3):589–607, 2008.

[21] Wenxin Jiang and Martin A Tanner. Gibbs posterior for variable selection in High-Dimensional classification and data mining. *Annals of statistics*, 36(5):2207–2231, 2008.

[22] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378, 2007.

[23] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[24] Jan-Matthis Lueckmann, Jan Boelts, David Greenberg, Pedro Goncalves, and Jakob Macke. Benchmarking simulation-based inference. In *International Conference on Artificial Intelligence and Statistics*, pages 343–351. PMLR, 2021.

[25] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. *arXiv preprint arXiv:1610.06545*, 2016.

[26] Jan-Matthis Lueckmann, Giacomo Bassetto, Theofanis Karaletsos, and Jakob H Macke. Likelihood-free inference with emulator networks. In *Symposium on Advances in Approximate Bayesian Inference*, pages 32–53. PMLR, 2019.

[27] Scott A Sisson and Yanan Fan. Likelihood-free markov chain monte carlo. *arXiv preprint arXiv:1001.2058*, 2010.

[28] Peter Grünwald and John Langford. Suboptimal behavior of bayes and mdl in classification under misspecification. *Machine Learning*, 66:119–149, 2007.

[29] Peter Grünwald and Thijs Van Ommen. Inconsistency of bayesian inference for misspecified linear models, and a proposal for repairing it. *Bayesian Analysis*, 2017.

[30] Patrick Cannon, Daniel Ward, and Sebastian M Schmon. Investigating the impact of model misspecification in neural simulation-based inference. *arXiv preprint arXiv:2209.01845*, 2022.

[31] Daniel Ward, Patrick Cannon, Mark Beaumont, Matteo Fasiolo, and Sebastian Schmon. Robust neural posterior estimation and statistical model criticism. *Advances in Neural Information Processing Systems*, 35:33845–33859, 2022.

[32] Allen Institute for Brain Science. Allen cell types database. *http://celltypes.brain-map.org/*, 2016.

[33] Corinne Teeter, Ramakrishnan Iyer, Vilas Menon, Nathan Gouwens, David Feng, Jim Berg, Aaron Szafer, Nicholas Cain, Hongkui Zeng, Michael Hawrylycz, Christof Koch, and Stefan Mihalas. Generalized leaky integrate-and-fire models classify multiple neuron types. *Nature communications*, 9(1):709, February 2018.

[34] Martin Pospischil, Maria Toledo-Rodriguez, Cyril Monier, Zuzanna Piwkowska, Thierry Bal, Yves Frégnac, Henry Markram, and Alain Destexhe. Minimal hodgkin–huxley type models for different classes of cortical and thalamic neurons. *Biological cybernetics*, 99:427–441, 2008.

[35] Romain Brette. What is the most realistic single-compartment model of spike initiation? *PLoS computational biology*, 11(4):e1004114, 2015.

[36] Nicholas Tolley, Pedro LC Rodrigues, Alexandre Gramfort, and Stephanie R Jones. Methods and considerations for estimating parameters in biophysically detailed neural models with simulation based inference. *bioRxiv*, pages 2023–04, 2023.

[37] Yves Bernaerts, Michael Deistler, Pedro J Goncalves, Jonas Beck, Marcel Stimberg, Federico Scala, Andreas S Tolias, Jakob H Macke, Dmitry Kobak, and Philipp Berens. Combined statistical-mechanistic modeling links ion channel genes to physiology of cortical neuron types. *bioRxiv*, pages 2023–03, 2023.

[38] Joshua H Goldwyn and Eric Shea-Brown. The what and where of adding channel noise to the hodgkin-huxley equations. *PLoS computational biology*, 7(11):e1002247, 2011.

[39] Richard David Wilkinson. Approximate bayesian computation (ABC) gives exact results under the assumption of model error. *Statistical applications in genetics and molecular biology*, 12(2):129–141, May 2013.

[40] Michael U Gutmann and Jukka Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research*, 2016.

[41] Volodimir Begy and Erich Schikuta. Error-guided likelihood-free MCMC. *arXiv*, October 2020.

[42] Daniel Ward, Patrick Cannon, Mark Beaumont, Matteo Fasiolo, and Sebastian M Schmon. Robust neural posterior estimation and statistical model criticism. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[43] Ryan P Kelly, David J Nott, David T Frazier, David J Warne, and Chris Drovandi. Misspecification-robust sequential neural likelihood. *arXiv preprint arXiv:2301.13368*, 2023.

[44] Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Morteza Ibrahimi, Xiuyuan Lu, and Benjamin Van Roy. Epistemic neural networks. July 2021.

[45] Salem Lahlou, Moksh Jain, Hadi Nekoei, Victor I Butoi, Paul Bertin, Jarrid Rector-Brooks, Maksym Korablyov, and Yoshua Bengio. DEUP: Direct epistemic uncertainty prediction. October 2022.

[46] Radford M Neal. Slice sampling. *The annals of statistics*, 31(3):705–767, 2003.

[47] Samuel Wiqvist, Jes Frellsen, and Umberto Picchini. Sequential neural posterior and likelihood approximation. *arXiv preprint arXiv:2102.06522*, 2021.

[48] Manuel Glöckler, Michael Deistler, and Jakob H. Macke. Variational methods for simulation-based inference. In *International Conference on Learning Representations*, 2022.

[49] Iain Murray, Zoubin Ghahramani, and David MacKay. MCMC for doubly-intractable distributions. June 2012.

[50] Pei-Shien Wu and Ryan Martin. A comparison of learning rate selection methods in generalized bayesian inference. *Bayesian Analysis*, 18(1):105–132, 2023.

[51] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[52] Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019.

[53] Scott A Sisson, Yanan Fan, and Mark M Tanaka. Sequential monte carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765, 2007.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[55] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[56] Alvaro Tejero-Cantero, Jan Boelts, Michael Deistler, Jan-Matthis Lueckmann, Conor Durkan, Pedro J. Gonçalves, David S. Greenberg, and Jakob H. Macke. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52):2505, 2020.

[57] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *Advances in neural information processing systems*, 32, 2019.

[58] Michael Deistler, Pedro J. Goncalves, and Jakob H. Macke. Truncated proposals for scalable and hassle-free simulation-based inference. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

Appendix

A1 Software and Data

We used PyTorch for all neural networks [51] and hydra to track all configurations [52]. Code to reproduce results is available at `https://github.com/mackelab/neuralgbi`.

A2 Common cost functions as expectations of the likelihood

As described in Sec. 3, our framework includes all cost functions which can be written as expectations over the likelihood: $\ell(\boldsymbol{\theta}; \mathbf{x}_o) = \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}, \mathbf{x}_o)]$. Below, we demonstrate that the average mean-squared error (MSE), the maximum mean discrepancy (MMD^2) [23], and the energy score [22] fall into this category.

Average Mean-squared error The average mean-squared error between samples from the likelihood $p(\mathbf{x}|\boldsymbol{\theta})$ and an observation $\mathbf{x}_o$ is defined as:

$$\text{MSE}(p(\mathbf{x}|\boldsymbol{\theta}), \mathbf{x}_o) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\boldsymbol{\theta})}[||\mathbf{x} - \mathbf{x}_o||^2]$$

which is trivially of the form required for our framework.

Maximum mean discrepancy Throughout this paper, we use the maximum mean discrepancy *squared* (MMD^2) [23]. The MMD^2 between the likelihood $p(\mathbf{x}|\boldsymbol{\theta})$ and the data distribution $p(\mathbf{x}_o)$ is defined as:

$$\begin{aligned}&\text{MMD}^2(p(\mathbf{x}|\boldsymbol{\theta}), p(\mathbf{x}_o)) = \\ &\mathbb{E}_{\mathbf{x},\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[k(\mathbf{x}, \mathbf{x}')] + \mathbb{E}_{\mathbf{x}_o,\mathbf{x}'_o\sim p(\mathbf{x}_o)}[k(\mathbf{x}_o, \mathbf{x}'_o)] + \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\boldsymbol{\theta}),\mathbf{x}_o\sim p(\mathbf{x}_o)}[k(\mathbf{x}, \mathbf{x}_o)]\end{aligned}$$

Assume we are given K iid samples as observations $\mathbf{x}_o^{(1,\dots,K)}$. Thus, the MMD^2 becomes:

$$\begin{aligned}&\text{MMD}^2(p(\mathbf{x}|\boldsymbol{\theta}), p(\mathbf{x}_o)) = \\ &\mathbb{E}_{\mathbf{x},\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[k(\mathbf{x}, \mathbf{x}')] + \frac{1}{K^2}\sum_{i=1}^{K}\sum_{j=1}^{K} k(\mathbf{x}_o^i, \mathbf{x}_o^j) + \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\boldsymbol{\theta})}\frac{1}{K}\sum_{i}^{K} k(\mathbf{x}, \mathbf{x}_o^i)\end{aligned}$$

which can be written as a single expectation over $p(\mathbf{x}|\boldsymbol{\theta})$

$$\begin{aligned}&\text{MMD}^2(p(\mathbf{x}|\boldsymbol{\theta}), p(\mathbf{x}_o)) = \\ &\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\boldsymbol{\theta})}\Big[\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[k(\mathbf{x}, \mathbf{x}')] + \frac{1}{K^2}\sum_{i=1}^{K}\sum_{j=1}^{K} k(\mathbf{x}_o^i, \mathbf{x}_o^j) + \frac{1}{K}\sum_{i}^{K} k(\mathbf{x}, \mathbf{x}_o^i)\Big]\end{aligned}$$

and follows the form required for our framework.

Energy score Following Gneiting et al. [22], the (negative, in order to fit our notation) energy score (ES) is defined as

$$\text{ES}(p(\mathbf{x}|\boldsymbol{\theta})|\mathbf{x}_o) = -\frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[||\mathbf{x} - \mathbf{x}'||^\beta] + \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\boldsymbol{\theta})}[||\mathbf{x} - \mathbf{x}_o||^\beta].$$

This can straight-forwardly be written as a single expectation over the likelihood

$$\text{ES}(p(\mathbf{x}|\boldsymbol{\theta})|\mathbf{x}_o) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\boldsymbol{\theta})}[\frac{1}{2}\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[||\mathbf{x} - \mathbf{x}'||^\beta] - ||\mathbf{x} - \mathbf{x}_o||^\beta]$$

and, thus, falls within our framework.

A3 Convergence proofs

The proofs closely follow standard proofs that regression converges to the conditional expectation.

A3.1 Proposition 1 and proof

Proposition 1. *Let $p(\boldsymbol{\theta}, \mathbf{x})$ be the joint distribution over parameters and data. Let $\ell(\boldsymbol{\theta}; \mathbf{x}_o)$ be a cost function that can be written as $\ell(\boldsymbol{\theta}; \mathbf{x}_o) = \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}, \mathbf{x}_o)] = \int_{\mathbf{x}} d(\mathbf{x}, \mathbf{x}_o) p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x}$ and let $f_\phi(\cdot)$ be a function parameterized by ϕ. Then, the loss function $\mathcal{L} = \mathbb{E}_{p(\boldsymbol{\theta},\mathbf{x})}[(f_\phi(\boldsymbol{\theta}) - d(\mathbf{x}, \mathbf{x}_o))^2]$ is minimized if and only if, for all $\boldsymbol{\theta} \in supp(p(\boldsymbol{\theta}))$, $f_\phi(\boldsymbol{\theta}) = \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}, \mathbf{x}_o)]$.*

Proof. We aim to prove that

$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - g(\boldsymbol{\theta}))^2] \geq \mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)])^2]$$

for every function $g(\boldsymbol{\theta})$. We begin by rearranging terms:

$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - g(\boldsymbol{\theta}))^2] =$$
$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)] + \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)] - g(\boldsymbol{\theta}))^2] =$$
$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)])^2 + (\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)] - g(\boldsymbol{\theta}))^2] + X$$

with

$$X = \mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)])(\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)] - g(\boldsymbol{\theta}))].$$

By the law of iterated expectations, one can show that $X = 0$:

$$X = \mathbb{E}_{\boldsymbol{\theta}'\sim p(\boldsymbol{\theta})}\mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta})}\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\boldsymbol{\theta})}[(d(\mathbf{x}, \mathbf{x}_o) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)])(\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)] - g(\boldsymbol{\theta}))].$$

The first term in the products reads a difference of the same term, and, thus, is zero.

Thus, since $X = 0$ and since $\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)] - g(\boldsymbol{\theta})^2 \geq 0$, we have

$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - g(\boldsymbol{\theta}))^2] \geq \mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)])^2].$$

□

A3.2 Proposition 2 and proof

Proposition 2. *Let $p(\boldsymbol{\theta}, \mathbf{x})$ be the joint distribution over parameters and data and let $p(\mathbf{x}_t)$ be a distribution of target samples. Let $\ell(\boldsymbol{\theta}; \mathbf{x}_o)$ be a cost function and $f_\phi(\cdot)$ a parameterized function as in proposition 1. Then, the loss function $\mathcal{L} = \mathbb{E}_{p(\boldsymbol{\theta},\mathbf{x})p(\mathbf{x}_t)}[(f_\phi(\boldsymbol{\theta}, \mathbf{x}_t) - d(\mathbf{x}, \mathbf{x}_t))^2]$ is minimized if and only if, for all $\boldsymbol{\theta} \in supp(p(\boldsymbol{\theta}))$ and all $\mathbf{x}_t \in supp(p(\mathbf{x}_t))$ we have $f_\phi(\boldsymbol{\theta}, \mathbf{x}_t) = \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}, \mathbf{x}_t)]$.*

Proof. We aim to prove that

$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x}),\mathbf{x}_t\sim p(\mathbf{x}_t)}[(d(\mathbf{x}, \mathbf{x}_t) - g(\boldsymbol{\theta}, \mathbf{x}_t))^2] \geq$$
$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x}),\mathbf{x}_t\sim p(\mathbf{x}_t)}[(d(\mathbf{x}, \mathbf{x}_t) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_t)])^2]$$

for every function $g(\boldsymbol{\theta}, \mathbf{x}_t)$. We begin as in proposition 1:

$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x}),\mathbf{x}_t\sim p(\mathbf{x}_t)}[(d(\mathbf{x}, \mathbf{x}_t) - g(\boldsymbol{\theta}, \mathbf{x}_t))^2] =$$
$$\mathbb{E}_{p(\mathbf{x}_t)}[\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_t) - g(\boldsymbol{\theta}, \mathbf{x}_t))^2]$$

Below, we prove that, for *any* $\mathbf{x}_t$, the optimal $g(\boldsymbol{\theta}, \mathbf{x}_t)$ is the conditional expectation $\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_t)]$:

$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_t) - g(\boldsymbol{\theta}, \mathbf{x}_t))^2] =$$
$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_t) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_t)] + \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_t)] - g(\boldsymbol{\theta}, \mathbf{x}_t))^2] =$$
$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_t) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_t)])^2 + (\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_t)] - g(\boldsymbol{\theta}, \mathbf{x}_t))^2] + X$$

with

$$X = \mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)])(\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)] - g(\boldsymbol{\theta}, \mathbf{x}_t))],$$

which, as above, is $X = 0$ (proof is identical to proposition 1). Thus, since $\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_t)] - g(\boldsymbol{\theta}, \mathbf{x}_t)^2 \geq 0$, we have:

$$\mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_t) - g(\boldsymbol{\theta}, \mathbf{x}_t))^2] \geq \mathbb{E}_{\theta,\mathbf{x}\sim p(\boldsymbol{\theta},\mathbf{x})}[(d(\mathbf{x}, \mathbf{x}_o) - \mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)])^2]$$

Because this inequality holds for *any* $\mathbf{x}_t$, the average over $p(\mathbf{x}_t)$ will also be minimized if and only if $g(\boldsymbol{\theta}, \mathbf{x}_t))$ matches the conditional expectation $\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x}|\boldsymbol{\theta})}[d(\mathbf{x}', \mathbf{x}_o)]$ for any $\mathbf{x}_t$ within the support of $p(\mathbf{x}_t)$. □

A4 Further experimental details

A4.1 Details on training procedure

Dataset construction We generate samples from the prior $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta})$ and corresponding simulations $\mathbf{x}_i \sim p(\mathbf{x}|\boldsymbol{\theta}_i)$, to obtain parameter-simulation pairs, $\Theta, X = \{\boldsymbol{\theta}_i, \mathbf{x}_i\}_{i=1\ldots N}$. Next, a dataset of target data points, X_{target}, is constructed by concatenating in the batch-dimension: 1) X, 2) a random subset of X augmented with Gaussian noise with a specified variance, i.e., $\mathbf{x}_i + \boldsymbol{\epsilon}_i, \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \sigma^2\mathbf{I})$, and 3) any number of real experimental observations the user wishes to include, X_{real}; in total, $N_{\text{target}} = N + N_{\text{noised}} + N_{\text{real}}$. Note that X_{target} does not technically need to include any simulated or real data, only 'realistic' targets in the neighborhood of the observed data that one eventually wants to perform inference for. Practically, 1) has already been simulated, and 2) and 3) does not require additional simulations while improving performance through noise augmentation and having access to real data targets. Finally, a pairwise distance matrix D is computed with elements $d_{i,j} = d(\mathbf{x}_i, \mathbf{x}_t), \mathbf{x}_i \in X, \mathbf{x}_t \in X_{\text{target}}$. D can either be pre-computed in full, or partially within the training loop.

Network optimization and convergence to GBI loss Given dataset $\Theta, X, X_{\text{target}}, D$, a fully connected feed-forward deep neural network with weights ϕ is trained to minimize the mean squared error loss: $\frac{1}{N_{\text{sim}} n_{\text{targets}}} \sum_{i=1}^{N_{\text{sim}}} \sum_{t=1}^{n_{\text{target}}} (\text{NN}_\phi(\boldsymbol{\theta}_i, \mathbf{x}_t) - d(\mathbf{x}_i, \mathbf{x}_t))^2$. In other words, for a parameter configuration $\boldsymbol{\theta}_i$ and a target data point $\mathbf{x}_t$, the network is trained to predict the distance $d_{i,t}$ between the corresponding single (stochastic) simulation $\mathbf{x}_i$ and the target $\mathbf{x}_t$. In every training epoch, n_{target} target simulations (usually 1-10, compared to $N_{\text{target}} > 10000$) are randomly sub-sampled per $\boldsymbol{\theta}_i$, drastically reducing training time and allowing the relevant $d_{i,t}$ to be computed on the fly.

Note that we do not wish to accurately predict the distance $d(\mathbf{x}_i, \mathbf{x}_t)$ in data-space for individual simulations $\mathbf{x}_i$, but rather the loss $\ell(\boldsymbol{\theta}_i; \mathbf{x}_t)$. Conveniently, with mean squared error as the objective function for network training, for a fixed pair of $\boldsymbol{\theta}_i, \mathbf{x}_t$, the trained network predicts the mean distance, $\mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta}_i)}[d(\mathbf{x}, \mathbf{x}_t)]$, precisely the loss function we target in Eq.2 (proof in Appendix).

A4.2 Description of benchmark tasks, distance and cost functions

A4.2.1 Uniform 1D

A one-dimensional task with uniform noise to illustrate how expected distance from observation can be used as a cost function for inference, especially in the case of model misspecification:

Prior	$\mathcal{U}(-1.5, 1.5)$
Simulator	$\mathbf{x}\|\boldsymbol{\theta} = g(z) + \epsilon$, where $\epsilon \sim \mathcal{U}(-0.25, 0.25), z = 0.8 \times (\boldsymbol{\theta} + 0.25)$, and $x_{\text{noiseless}} = g(z) = 0.1627 + 0.9073z - 1.2197z^2 - 1.4639z^3 + 1.4381z^4$.
Dimensionality	$\boldsymbol{\theta} \in \mathbb{R}^1, \mathbf{x} \in \mathbb{R}^1$
Cost function	$d(\mathbf{x}, \mathbf{x}_o)$: MSE; $p(\mathbf{x}\|\boldsymbol{\theta})$ computed exactly given uniform noise. For obtaining the true cost, Eq. 2 is analytically integrated over $x_{\text{noiseless}} \pm 0.25$
Posterior	Ground-truth GBI posterior samples are obtained via rejection sampling. We used the prior as proposal distribution.

A4.2.2 Two Moons

A two-dimensional task with a posterior that exhibits both global (bimodality) and local (crescent shape) structure to illustrate how algorithms deal with multimodality:

Prior	$\mathcal{U}(-\mathbf{1}, \mathbf{1})$
Simulator	$\boldsymbol{x}\|\boldsymbol{\theta} = \begin{bmatrix} r\cos(\alpha) + 0.25 \\ r\sin(\alpha) \end{bmatrix} + \begin{bmatrix} -\|\theta_1 + \theta_2\|/\sqrt{2} \\ (-\theta_1 + \theta_2)/\sqrt{2} \end{bmatrix}$, where $\alpha \sim \mathcal{U}(-\pi/2, \pi/2)$, and $r \sim \mathcal{N}(0.1, 0.01^2)$
Dimensionality	$\boldsymbol{\theta} \in \mathbb{R}^2, \mathbf{x} \in \mathbb{R}^2$
Cost function	$d(\mathbf{x}, \mathbf{x}_o)$: MSE; and $p(\mathbf{x}\|\boldsymbol{\theta})$ computed exactly. To obtain the true cost, Eq. 2 is numerically integrated over a 2D grid of $d\mathbf{x}$ with 500 equal bins in both dimensions, where $x^{(1)} \in [-1.2, 0.4], x^{(2)} \in [-1.6, 1.6]$.
Posterior	Ground-truth GBI posterior samples are obtained via rejection sampling with the prior as proposal distribution.
References	[24, 8]

A4.2.3 Linear Gaussian

Inference of the mean of a 10-d Gaussian model, in which the covariance is fixed. The (conjugate) prior is Gaussian:

Prior	$\mathcal{N}(\mathbf{0}, 0.1 \odot \mathbf{I})$
Simulator	$\mathbf{x}\|\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{x}\|\mathbf{m}_{\boldsymbol{\theta}} = \boldsymbol{\theta}, \mathbf{S} = 0.1 \odot \mathbf{I})$
Dimensionality	$\boldsymbol{\theta} \in \mathbb{R}^{10}, \mathbf{x} \in \mathbb{R}^{10}$
Cost function	$d(\mathbf{x}, \mathbf{x}_o)$: MSE; and $p(\mathbf{x}\|\boldsymbol{\theta})$ computed exactly. The true cost (Eq. 2) can be computed analytically.
Posterior	Ground-truth GBI posterior samples are obtained following the procedure in Lueckmann et al. [24]: We first ran MCMC to generate 10k samples from the GBI posterior. We then trained a neural spline flow on these 10k samples and, finally, used the trained flow as proposal distribution for rejection sampling.
References	[24, 8]

A4.2.4 Gaussian Mixture

The single-trial version of this task is common in the ABC literature. It consists of inferring the common mean of a mixture of two two-dimensional Gaussian distributions, one with much broader covariance than the other. In this study, we used this task to infer the common mean of the two distributions from five i.i.d. simulations:

Prior	$\mathcal{U}(-\mathbf{10}, \mathbf{10})$
Simulator	$\mathbf{x}\|\boldsymbol{\theta} \sim 0.5\,\mathcal{N}(\mathbf{x}\|\mathbf{m}_{\boldsymbol{\theta}} = \boldsymbol{\theta}, \mathbf{S} = \mathbf{I}) + 0.5\,\mathcal{N}(\mathbf{x}\|\mathbf{m}_{\boldsymbol{\theta}} = \boldsymbol{\theta}, \mathbf{S} = 0.01 \odot \mathbf{I})$
Dimensionality	$\boldsymbol{\theta} \in \mathbb{R}^2, \mathbf{x} \in \mathbb{R}^2$
Cost function	$d(\mathbf{x}, \mathbf{x}_o)$: MMD^2. The true cost (Eq. 2) is obtained by integrating the distance function on a grid for every trial independently and multiplying over the trials.
Posterior	Ground-truth GBI posterior samples are obtained via rejection sampling. As proposal, we used a Normal distribution centered around the ground truth parameter and with variance $\frac{50}{\beta}$.
References	[24, 53]

A4.3 Training, inference, and evaluation for benchmark tasks

ACE For the benchmark tasks, the cost estimation network is a residual network with 3 hidden layers of 64 units [54]. The training dataset was split 90:10 into training and validation sets, with $n_{\text{target}} = 2$ and 5 $\mathbf{x}_t$ randomly sampled each epoch for evaluating training and validation loss, respectively. We used a batchsize of 500, i.e., 500 $\boldsymbol{\theta}$, 2 $\mathbf{x}_t$, for 1000 cost targets per training batch. Networks usually converge within 500 epochs, with 100 epochs of non-decreasing validation loss as the convergence criterion. For inference, we ran multi-chain slice sampling with 100 chains to sample the potential function. For the Gaussian Mixture task with 5 i.d.d. samples per simulation/observation, we appended to the cost estimation network a fully connected, permutation-invariant embedding network [55] (2 layers, 100 units each), which preprocessed the i.d.d. datapoints into a single 20-dimensional vector (but was trained end-to-end).

Kernel ABC We compared our algorithm with a version of kernel ABC on the benchmark tasks. For kernel ABC, we accepted samples (from the fixed set of 10000 prior simulations) with probability $\exp(-\beta \cdot d(\mathbf{x}, \mathbf{x}_o))$, where $d(\cdot, \cdot)$ is the distance function. In many cases, and especially for large β, this yielded very few or no accepted samples. We, therefore, reduced β until at least 50 samples were accepted.

NPE We used the implementation in the sbi toolbox [56], with neural spline flow [57] as density estimator (five transformation layers) to approximate the posterior. For tasks with bounded priors (all except Linear Gaussian), we appended a sigmoidal transformation to the flow such that its support matches the support of the prior [58]. For the Gaussian Mixture task, the same permutation-invariant embedding net was used as for ACE. Posterior samples were directly obtained from the trained flow. All other hyperparameters were the same as in the sbi toolbox, version 0.19.2 [56].

NLE and tempered NLE We used the implementation in the sbi toolbox [56], with neural spline flow [57] as density estimator (five transformation layers) to approximate the likelihood. For the Gaussian Mixture task with 5 i.d.d. simulations per sample/observation, they were split up as 5 independent simulations with the same $\boldsymbol{\theta}$ repeated 5 times, effectively having 5 times the training simulation budget. We ran multi-chain slice sampling with 100 chains to sample the potential function as the sum of log-prior probability and flow-approximated log-likelihood. Tempered NLE uses the same learned likelihood estimator, but the log-likelihood term in the potential function is scaled by β during MCMC sampling (same values as used for ACE). All other hyperparameters were the same as in the sbi toolbox, version 0.19.2 [56].

Noise augmentation For each task, we randomly subsampled 100 of all simulated datapoints ($\mathbf{x}$), and added Gaussian noise with zero-mean and standard deviation equaling two times the standard deviation of all prior predictives, i.e., $\mathcal{N}(\mathbf{0}, (2\sigma_x)^2\mathbf{I})$. We additionally varied noise amplitude $(0, 2\sigma_x, 5\sigma_x)$ and found little to no effect on benchmark performance (see Appendix A3).

Synthetic misspecified observations For the Uniform 1D, 2 Moons, and Linear Gaussian task, we generated an additional 100,000 prior predictive simulations and defined the prior predictive bound as the minimum and maximum (of each dimension) of those simulations and σ_x as their standard deviation. Then, to create 20 misspecified observations, we generated 20 model-simulations and added Gaussian noise with zero-mean and standard deviation equaling 0.5 σ_x (i.e., $\epsilon \sim \mathcal{N}(\mathbf{0}, (0.5\sigma_x)^2\mathbf{I})$) to the 20 observations iteratively until they are outside of the prior predictive bounds in every dimension. For the Gaussian Mixture task, we replaced the second Gaussian above by $\mathcal{N}(12.5 \times \text{sign}(\boldsymbol{\theta}), 0.5^2\mathbf{I})$, i.e. displacing the mean to the corner of the quadrant whose signs match $\boldsymbol{\theta}$, and slightly increasing the variance.

Metrics For the metrics in Fig. 3, in columns 1, 2, and 4 and all corresponding figures in the Appendix, results were aggregated across the 10 well-specified and misspecified samples separately, where marker and error bars represent mean and standard deviation over 10 observations. Columns 1 and 2 show average distance between observation and 5000 posterior predictive simulations from each algorithm (except ABC, for which there were 50 posterior samples). Column 3 shows, for each of the 10 well-specified observations and 3 β values, 3 random posterior samples for which we compare the ACE-estimated and ground-truth cost, i.e., $10 \times 3 \times 3 = 90$ points are shown for each

task. Column 4 shows C2ST score between 5000 ground-truth GBI posterior samples and ACE GBI posterior samples.

A4.4 Training procedure for Hodgkin-Huxley model

For NPE, we used the implementation in the sbi toolbox [56]. As density estimator, we used a neural spline flow [57] with five transformation layers. We used a batchsize of 5,000. We appended a sigmoidal transformation to the flow such that its support matches the support of the prior [58]. All other hyperparameters were the same as in the sbi toolbox, version 0.19.2 [56].

For ACE, we used a residual neural network [54] with 7 layers and 100 hidden units each. We used 10% of all simulations as held-out validation set and stopped training when the validation set did not decrease for 100 epochs. We used a batchsize of 5000. For sampling from the generalized posterior, we ran multi-chain slice sampling with 100 chains. As distance function, we used mean-absolute-error, where the distance in each summary statistic was z-scored with the standard deviation of the prior predictives. To generate $\mathbf{x}_t$, we used all simulations which generated between 5 and 40 spikes (since any reasonable experimental recording would fall within this range) and we appended 1000 augmented simulations to which we added Gaussian noise with two times the standard deviation of prior predictives that produce between 5 and 40 spikes. We did not use the experimental recordings from the Allen Cell Types database in $\mathbf{x}_t$.

A5 Hodgkin-Huxley model

We used the same model as Gonçalves et al. [10], which follows the model proposed in Pospischil et al. [34]. Briefly, the model contains four types of conductances (sodium, delayed-rectifier potassium, slow voltage-dependent potassium, and leak) and has a total of eight parameters that generate a time series which we reduce to seven summary statistics (spike count, mean resting potential, standard deviation of the resting potential, and the first four voltage moments mean, standard deviation, skew, curtosis, same as in Gonçalves et al. [10]).

A6 Supplementary figures

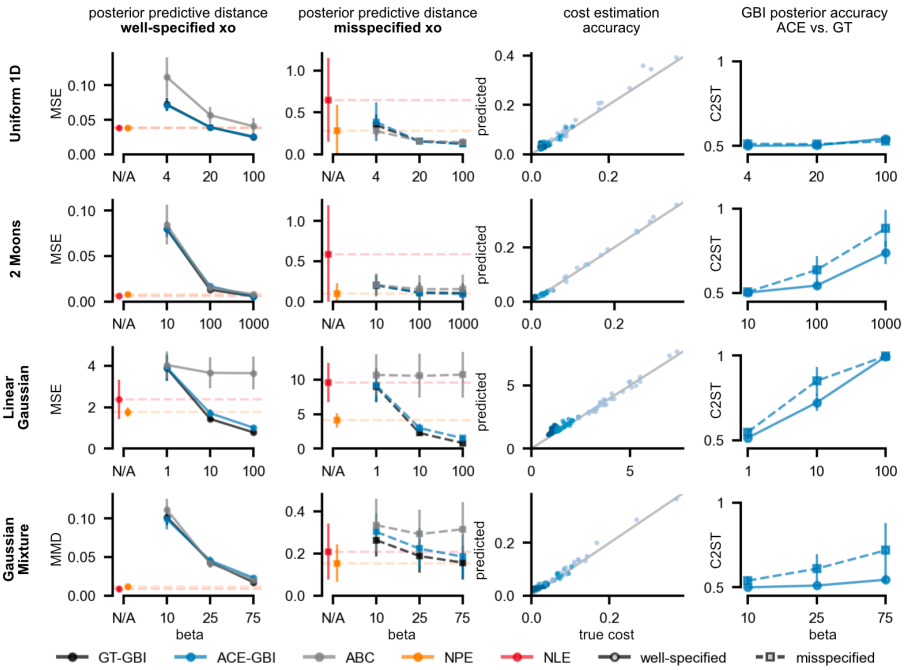


Figure A1: **Benchmark performance for 'seen' observations** Panels are the same as in Fig. 3, but for the 20 x_o that were included in the target dataset during training.

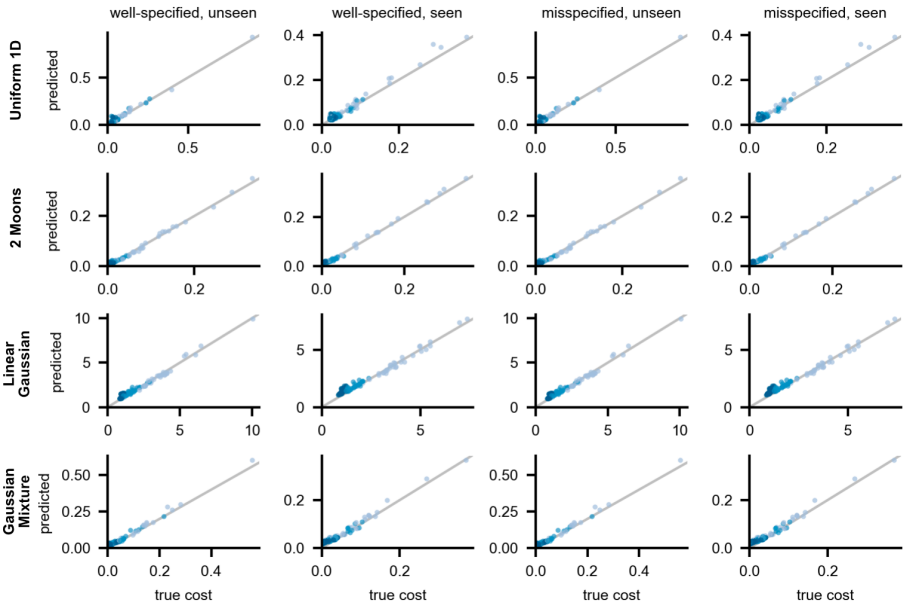


Figure A2: **Cost estimation accuracy for all observations** Columns are same as 3rd column in Fig. 3, but for all combinations of (unseen, seen) and (well-specified, misspecified) observations (10 each, 40 total) for each task. 1st column here is identical to Fig. 3 3rd column.

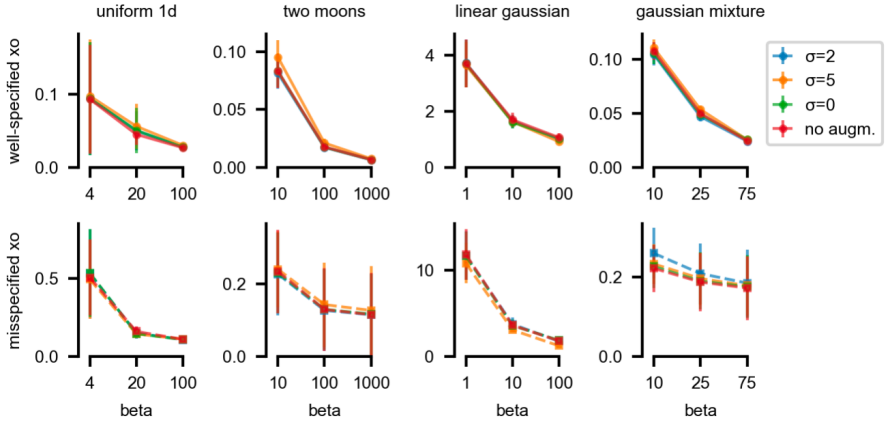


Figure A3: **Benchmark performance for ACE trained with various augmentation noise levels.** Results based on 10k training budget plus 100 noise augmented samples with varying noise amplitudes (blue, orange, green), as well as removing data augmentation altogether (i.e., no noise augmented simulated nor observed data, red). Overall, changing σ does not significantly impact performance compared to the original results ($2\sigma_x$, blue).

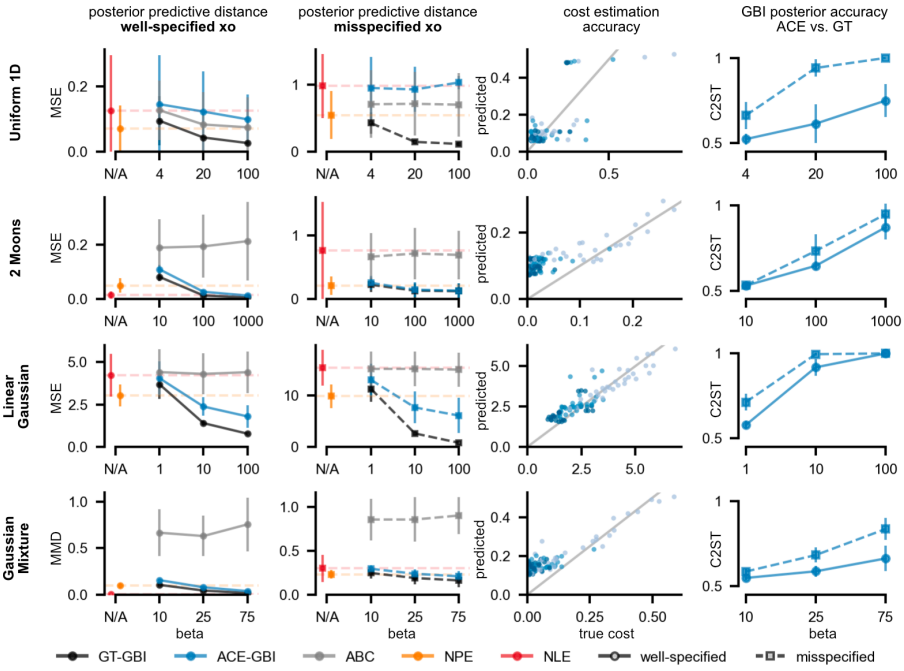


Figure A4: **Benchmark performance with simulation budget of 200** Panels are the same as in Fig. 3, but all algorithms trained with simulation budget of 200.

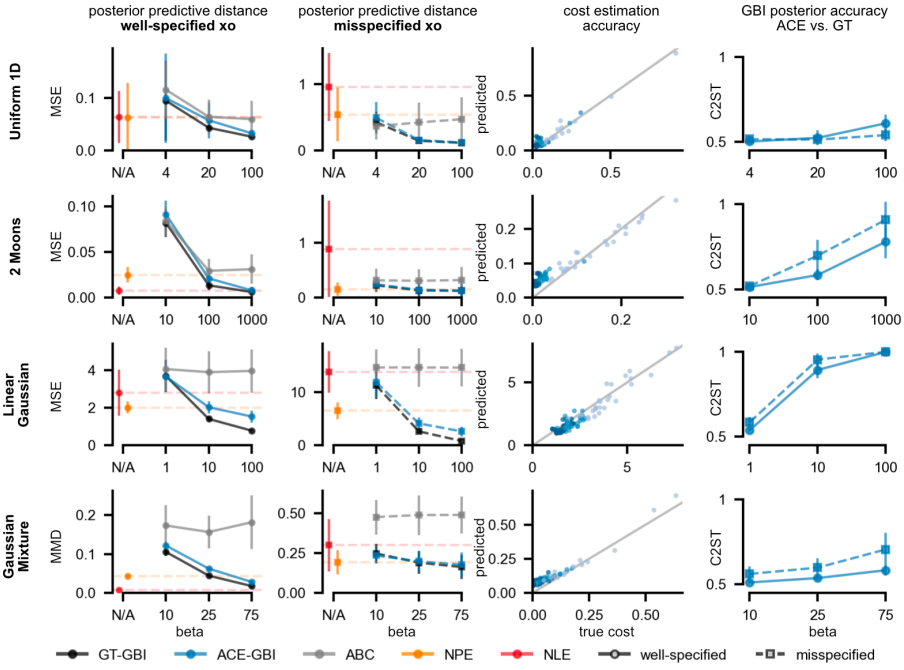


Figure A5: **Benchmark performance with simulation budget of 1000** Panels are the same as in Fig. 3, but all algorithms trained with simulation budget of 1000.

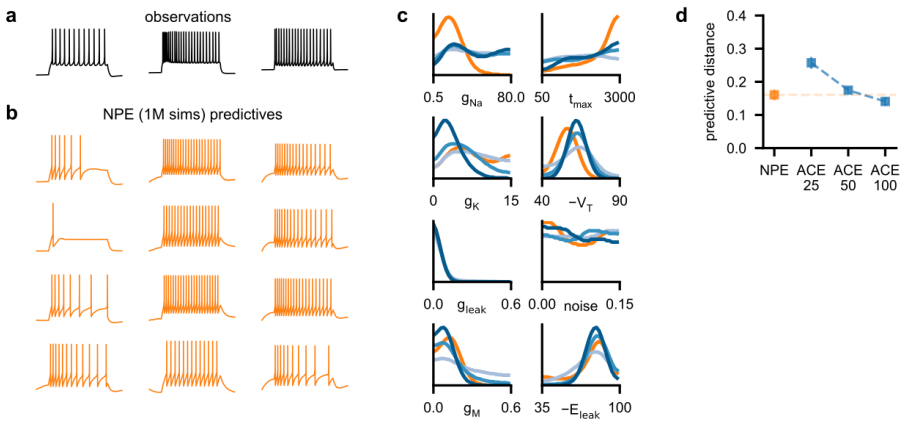


Figure A6: **NPE performance with 1 million simulations.** Panels are the same as in Fig. 4, but NPE was run with 1 million simulations. ACE is still run with 100K simulations and, thus, the ACE data is the same as in Fig. 4.

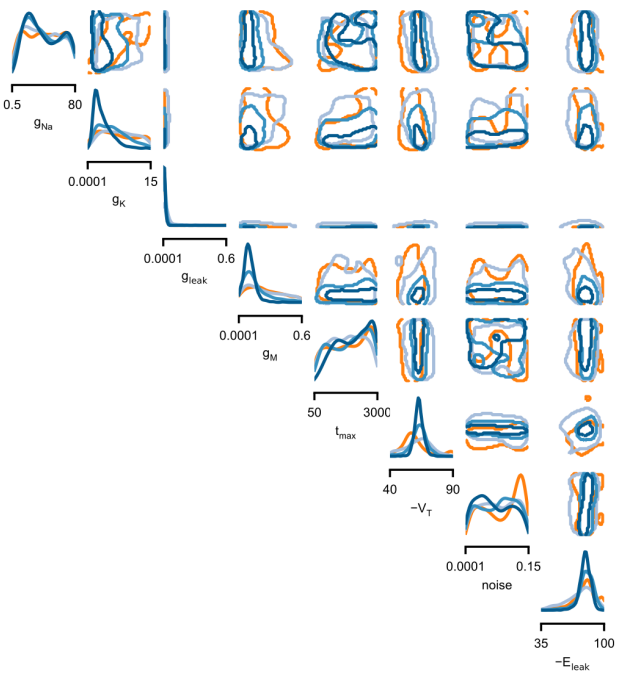


Figure A7: **Cornerplot of posterior distributions.** Diagonals are 1D-marginals, upper diagonals are 2D-marginals (68th-percentile contour from 5000 posterior samples). Orange: NPE with 100K simulations. Shades of blue: ACE with 100K simulations with $\beta = 25$ (light blue), $\beta = 50$ (medium blue), $\beta = 100$ (dark blue).

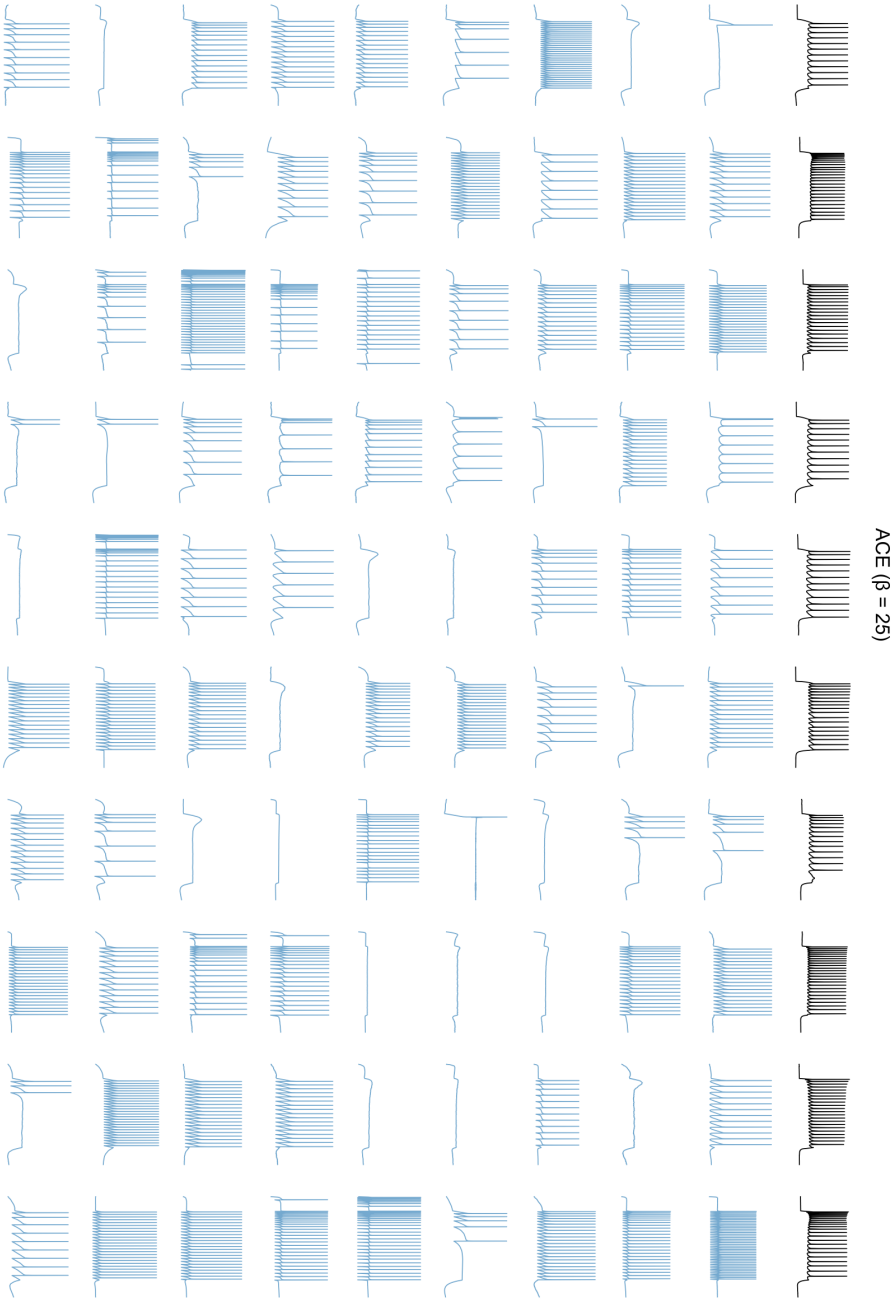


Figure A8: **Posterior predictive samples of ACE (with 100K simulations) with $\beta = 25$.** Top row (black): 10 experimental recordings from the Allen Cell Types database. Below: nine predictive samples given each of the ten observations.

Figure A9: **Posterior predictive samples of ACE (with 100K simulations) with $\beta = 50$.** Top row (black): 10 experimental recordings from the Allen Cell Types database. Below: Nine predictive samples given each of the ten observations.

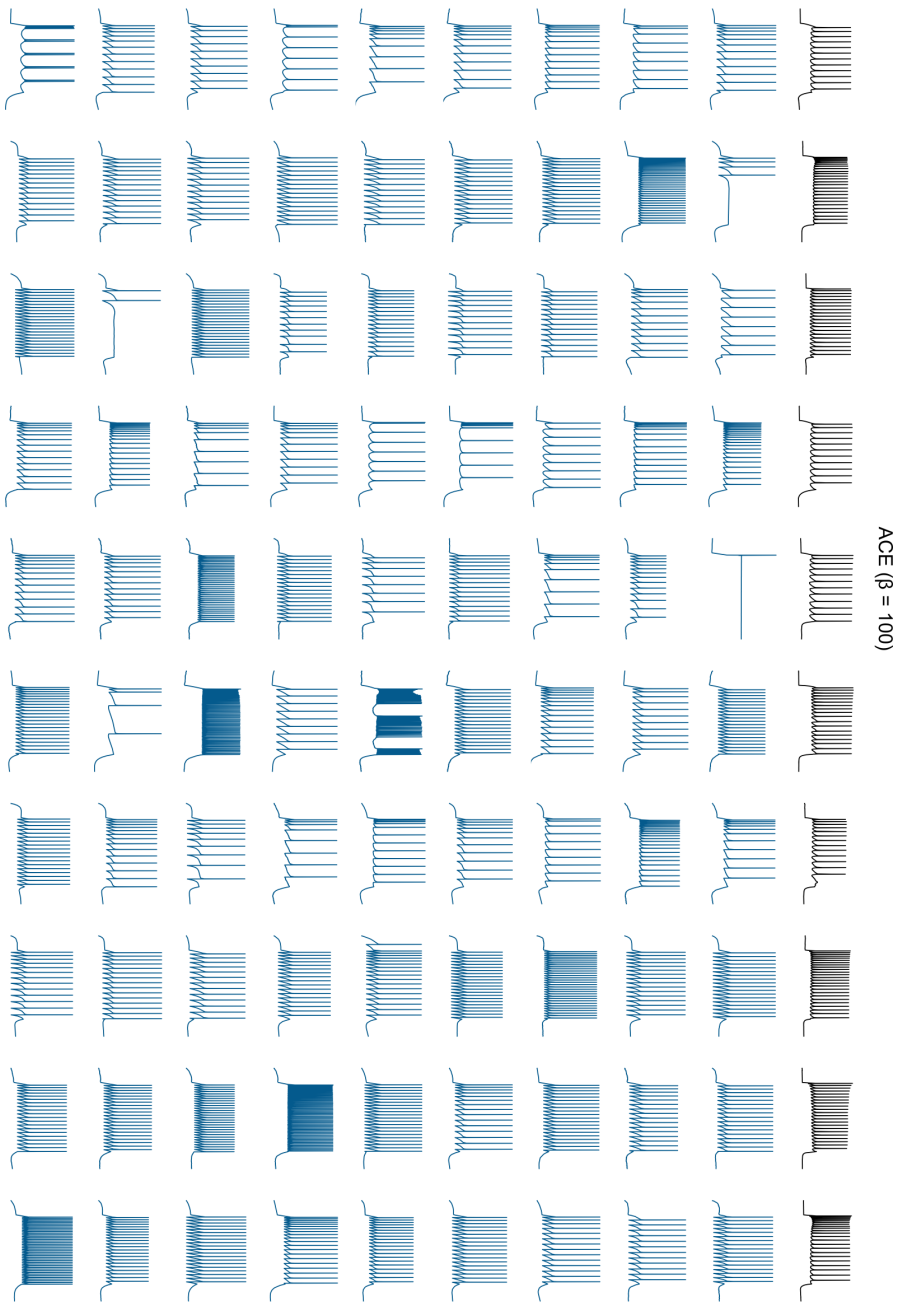


Figure A10: **Posterior predictive samples of ACE (with 100K simulations) with $\beta = 100$.** Top row (black): 10 experimental recordings from the Allen Cell Types database. Below: Nine predictive samples given each of the ten observations.

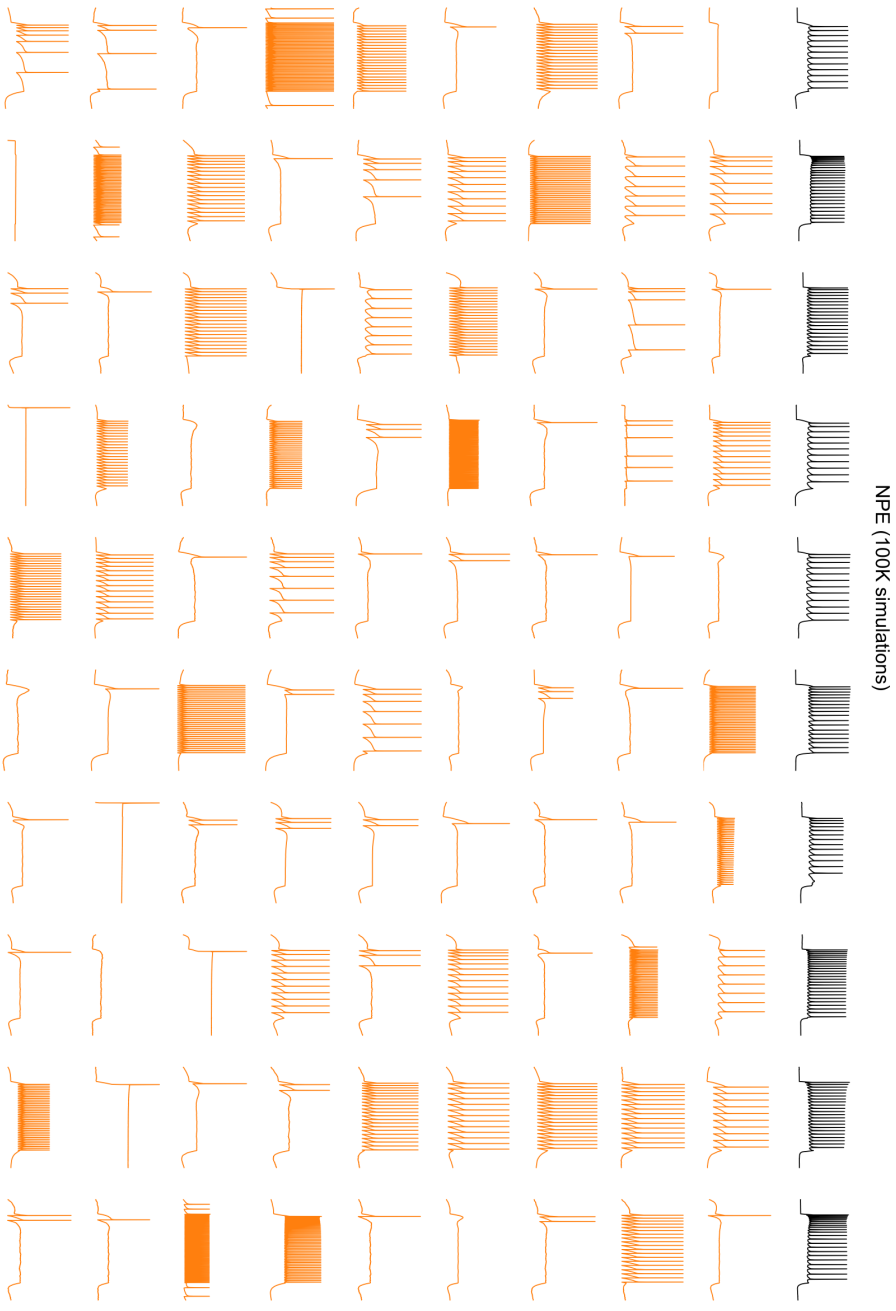


Figure A11: **Posterior predictive samples of NPE with 100K simulations.** Top row (black): 10 experimental recordings from the Allen Cell Types database. Below: Nine predictive samples given each of the ten observations.

Figure A12: **Posterior predictive samples of NPE with 1M simulations.** Top row (black): 10 experimental recordings from the Allen Cell Types database. Below: Nine predictive samples given each of the ten observations.

JOSS
The Journal of Open Source Software

sbi: A toolkit for simulation-based inference

Alvaro Tejero-Cantero[e, 1], **Jan Boelts**[e, 1], **Michael Deistler**[e, 1], **Jan-Matthis Lueckmann**[e, 1], **Conor Durkan**[e, 2], **Pedro J. Gonçalves**[1, 3], **David S. Greenberg**[1, 4], **and Jakob H. Macke**[1, 5, 6]

e Equally contributing authors **1** Computational Neuroengineering, Department of Electrical and Computer Engineering, Technical University of Munich **2** School of Informatics, University of Edinburgh **3** Neural Systems Analysis, Center of Advanced European Studies and Research (caesar), Bonn **4** Model-Driven Machine Learning, Centre for Materials and Coastal Research, Helmholtz-Zentrum Geesthacht **5** Machine Learning in Science, University of Tübingen **6** Empirical Inference, Max Planck Institute for Intelligent Systems, Tübingen

DOI: 10.21105/joss.02505

Software

- Review
- Repository
- Archive

Editor: Dan Foreman-Mackey
Reviewers:

- @EiffL
- @cranmer

Submitted: 14 July 2020
Published: 21 August 2020

Summary

Scientists and engineers employ stochastic numerical simulators to model empirically observed phenomena. In contrast to purely statistical models, simulators express scientific principles that provide powerful inductive biases, improve generalization to new data or scenarios and allow for fewer, more interpretable and domain-relevant parameters. Despite these advantages, tuning a simulator's parameters so that its outputs match data is challenging. Simulation-based inference (SBI) seeks to identify parameter sets that a) are compatible with prior knowledge and b) match empirical observations. Importantly, SBI does not seek to recover a single 'best' data-compatible parameter set, but rather to identify all high probability regions of parameter space that explain observed data, and thereby to quantify parameter uncertainty. In Bayesian terminology, SBI aims to retrieve the posterior distribution over the parameters of interest. In contrast to conventional Bayesian inference, SBI is also applicable when one can run model simulations, but no formula or algorithm exists for evaluating the probability of data given parameters, i.e. the likelihood.

We present `sbi`, a PyTorch-based package that implements SBI algorithms based on neural networks. `sbi` facilitates inference on black-box simulators for practising scientists and engineers by providing a unified interface to state-of-the-art algorithms together with documentation and tutorials.

Motivation

Bayesian inference is a principled approach for determining parameters consistent with empirical observations: Given a prior over parameters, a stochastic simulator, and observations, it returns a posterior distribution. In cases where the simulator likelihood *can* be evaluated, many methods for approximate Bayesian inference exist (e.g., Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953; Baydin et al., 2019; Graham & Storkey, 2017; Le, Baydin, & Wood, 2017; Neal, 2003). For more general simulators, however, evaluating the likelihood of data given parameters might be computationally intractable. Traditional algorithms for this 'likelihood-free' setting (Cranmer, Brehmer, & Louppe, 2020) are based on Monte-Carlo rejection (Pritchard, Seielstad, Perez-Lezaun, & Feldman, 1999; Sisson, Fan, & Tanaka, 2007), an approach known as *Approximate Bayesian Computation* (ABC). More recently, algorithms based on neural networks have been developed (Greenberg, Nonnenmacher, & Macke, 2019; Hermans, Begy, & Louppe, 2020; Lueckmann et al., 2017; Papamakarios & Murray, 2016;

The Journal of Open Source Software

Papamakarios, Sterratt, & Murray, 2019). These algorithms are not based on rejecting simulations, but rather train deep neural conditional density estimators or classifiers on simulated data. To aid in effective application of these algorithms to a wide range of problems, `sbi` closely integrates with PyTorch and offers state-of-the-art neural network-based SBI algorithms (Greenberg et al., 2019; Hermans et al., 2020; Papamakarios et al., 2019) with flexible choice of network architectures and flow-based density estimators. With `sbi`, researchers can easily implement new neural inference algorithms, benefiting from the infrastructure to manage simulators and a unified posterior representation. Users, in turn, can profit from a single inference interface that allows them to either use their own custom neural network, or choose from a growing library of preconfigured options provided with the package.

Related software and use in research

We are aware of several mature packages that implement SBI algorithms. `elfi` (Lintusaari et al., 2018) is a package offering BOLFI, a Gaussian process-based algorithm (Gutmann & Corander, 2016), and some classical ABC algorithms. The package `carl` (Louppe, Cranmer, & Pavez, 2016) implements the algorithm described in Cranmer, Pavez, & Louppe (2015). Two other SBI packages, currently under development, are `hypothesis` (Hermans, 2019) and `pydelfi` (Alsing, 2019). `pyabc` (Klinger, Rickert, & Hasenauer, 2018) and `ABCpy` (Dutta, Schoengens, Onnela, & Mira, 2017) are two packages offering a diversity of ABC algorithms.

`sbi` is closely integrated with PyTorch (Paszke et al., 2019) and uses `nflows` (Durkan, Bekasov, Papamakarios, & Murray, 2019) for flow-based density estimators. `sbi` builds on experience accumulated developing `delfi` (mackelab.org, 2017), which it succeeds. `delfi` was based on `theano` (Al-Rfou et al., 2016) (development discontinued) and developed both for SBI research (Greenberg et al., 2019; Lueckmann et al., 2017) and for scientific applications (Gonçalves et al., 2019). The `sbi` codebase started as a fork of `lfi` (Durkan, 2020), developed for Durkan et al. (2020).

Description

`sbi` currently implements three families of neural inference algorithms:

- Sequential Neural *Posterior* Estimation (SNPE) trains a deep neural density estimator that directly estimates the posterior distribution of parameters given data. Afterwards, it can sample parameter sets from the posterior, or evaluate the posterior density on any parameter set. Currently, SNPE-C (Greenberg et al., 2019) is implemented in `sbi`.
- Sequential Neural *Likelihood* Estimation (SNLE) (Papamakarios et al., 2019) trains a deep neural density estimator of the likelihood, which then allows to sample from the posterior using e.g. MCMC.
- Sequential Neural *Ratio* Estimation (SNRE) (Durkan et al., 2020; Hermans et al., 2020) trains a classifier to estimate density ratios, which in turn can be used to sample from the posterior e.g. with MCMC.

The inference step returns a `NeuralPosterior` object that represents the uncertainty about the parameters conditional on an observation, i.e. the posterior distribution. This object can be sampled from —and if the chosen algorithm allows, evaluated— with the same API as a standard PyTorch probability distribution.

An important challenge in making SBI algorithms usable by a broader community is to deal with diverse, often pre-existing, complex simulators. `sbi` works with any simulator as long as it can be wrapped in a Python callable. Furthermore, `sbi` ensures that custom simulators

The Journal of Open Source Software

work well with neural networks, e.g. by performing automatic shape inference, standardizing inputs or handling failed simulations. To maximize simulator performance, `sbi` leverages vectorization where available and optionally parallelizes simulations using `joblib` (Varoquaux, 2008). Moreover, if dimensionality reduction of the simulator output is desired, `sbi` can use a trainable summarizing network to extract relevant features from raw simulator output and spare the user manual feature engineering.

In addition to the full-featured interface, `sbi` provides also a *simple* interface which consists of a single function call with reasonable defaults. This allows new users to get familiarized with simulation-based inference and quickly obtain results without having to define custom networks or tune hyperparameters.

With `sbi`, we aim to support scientific discovery and computational engineering by making Bayesian inference applicable to the widest class of models (simulators with no likelihood available), and practical for complex problems. We have designed an open architecture and adopted community-oriented development practices in order to invite other machine-learning researchers to join us in this long-term vision.

Acknowledgements

This work has been supported by the German Federal Ministry of Education and Research (BMBF, project 'ADIMEM', FKZ 01IS18052 A-D), the German Research Foundation (DFG) through SFB 1089 'Synaptic Microcircuits', SPP 2041 'Computational Connectomics' and Germany's Excellence Strategy – EXC-Number 2064/1 – Project number 390727645.

Conor Durkan was supported by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

We are grateful to Artur Bekasov, George Papamakarios and Iain Murray for making `nflows` (Durkan et al., 2019) available, a package for normalizing flow-based density estimation which `sbi` leverages extensively.

References

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., et al. (2016). Theano: A python framework for fast computation of mathematical expressions. *arXiv*, arXiv–1605.

Alsing, J. (2019). pydelfi: Density estimation likelihood-free inference. *GitHub repository*. https://github.com/justinalsing/pydelfi; GitHub.

Baydin, A. G., Shao, L., Bhimji, W., Heinrich, L., Meadows, L., Liu, J., Munk, A., et al. (2019). Etalumis: Bringing probabilistic programming to scientific simulators at scale. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 1–24). doi:10.1145/3295500.3356180

Cranmer, K., Brehmer, J., & Louppe, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*. doi:10.1073/pnas.1912789117

Cranmer, K., Pavez, J., & Louppe, G. (2015). Approximating likelihood ratios with calibrated discriminative classifiers. *arXiv preprint arXiv:1506.02169*.

Durkan, C. (2020). Lfi. *GitHub repository*. https://github.com/conormdurkan/lfi; GitHub.

Durkan, C., Bekasov, A., Papamakarios, G., & Murray, I. (2019). nflows: Normalizing flows in PyTorch. *GitHub repository*. https://github.com/bayesiains/nflows; GitHub.

The Journal of Open Source Software

Durkan, C., Murray, I., & Papamakarios, G. (2020). On contrastive learning for likelihood-free inference. *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of machine learning research, *98*.

Dutta, R., Schoengens, M., Onnela, J.-P., & Mira, A. (2017). ABCpy: A user-friendly, extensible, and parallel library for approximate bayesian computation. In *Proceedings of the platform for advanced scientific computing conference* (pp. 8:1–8:9). doi:10.1145/3093172.3093233

Gonçalves, P. J., Lueckmann, J.-M., Deistler, M., Nonnenmacher, M., Öcal, K., Bassetto, G., Chintaluri, C., et al. (2019). Training deep neural density estimators to identify mechanistic models of neural dynamics. *bioRxiv*, 838383. doi:10.1101/838383

Graham, M. M., & Storkey, A. J. (2017). Asymptotically exact inference in differentiable generative models. *Electronic Journal of Statistics*, *11*(2), 5105–5164. doi:10.1214/17-EJS1340SI

Greenberg, D., Nonnenmacher, M., & Macke, J. (2019). Automatic posterior transformation for likelihood-free inference. In *Proceedings of the 36th international conference on machine learning*, Proceedings of machine learning research (Vol. 97, pp. 2404–2414). PMLR.

Gutmann, M. U., & Corander, J. (2016). Bayesian optimization for likelihood-free inference of simulator-based statistical models. *The Journal of Machine Learning Research*, *17*(1), 4256–4302.

Hermans, J. (2019). Hypothesis. *GitHub repository*. https://github.com/montefiore-ai/hypothesis; GitHub.

Hermans, J., Begy, V., & Louppe, G. (2020). Likelihood-free MCMC with approximate likelihood ratios. In *Proceedings of the 37th international conference on machine learning*, Proceedings of machine learning research (Vol. 98). PMLR.

Klinger, E., Rickert, D., & Hasenauer, J. (2018). pyABC: Distributed, likelihood-free inference. *Bioinformatics*, *34*(20), 3591–3593. doi:10.1093/bioinformatics/bty361

Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference compilation and universal probabilistic programming. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017*, *54*.

Lintusaari, J., Vuollekoski, H., Kangasrääsiö, A., Skytén, K., Järvenpää, M., Marttinen, P., Gutmann, M. U., et al. (2018). ELFI: Engine for likelihood-free inference. *The Journal of Machine Learning Research*, *19*(1), 643–649.

Louppe, G., Cranmer, K., & Pavez, J. (2016). carl: A likelihood-free inference toolbox. *Journal of Open Source Software*, *1*(1), 11. doi:10.21105/joss.00011

Lueckmann, J.-M., Goncalves, P. J., Bassetto, G., Öcal, K., Nonnenmacher, M., & Macke, J. H. (2017). Flexible statistical inference for mechanistic models of neural dynamics. In *Advances in neural information processing systems 30* (pp. 1289–1299).

mackelab.org. (2017). DELFI: Density estimation likelihood-free inference. *GitHub repository*. https://github.com/mackelab/delfi; GitHub.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. doi:10.1063/1.1699114

Neal, R. M. (2003). Slice sampling. *The Annals of Statistics*, *31*(3), 705–741. doi:10.1214/aos/1056562461

Papamakarios, G., & Murray, I. (2016). Fast ϵ-free inference of simulation models with bayesian conditional density estimation. In *Advances in neural information processing systems 29* (pp. 1028–1036).

Papamakarios, G., Sterratt, D., & Murray, I. (2019). Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. *Proceedings of Machine Learning Research*, Proceedings of machine learning research, *89*, 837–848.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035).

Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., & Feldman, M. W. (1999). Population growth of human Y chromosomes: A study of Y chromosome microsatellites. *Molecular biology and evolution*, *16*(12), 1791–1798. doi:10.1093/oxfordjournals.molbev.a026091

Sisson, S. A., Fan, Y., & Tanaka, M. M. (2007). Sequential Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, *104*(6), 1760–1765. doi:10.1073/pnas.0607208104

Varoquaux, G. (2008). Joblib. *GitHub repository*. https://github.com/joblib/joblib; GitHub.

sbi reloaded: a toolkit for simulation-based inference workflows

Jan Boelts[*,†,1,2,3], **Michael Deistler**[*,†,1,2], **Manuel Gloeckler**[†,1,2], **Álvaro Tejero-Cantero**[†,3,4], **Jan-Matthis Lueckmann**[†,5], **Guy Moss**[†,1,2], **Peter Steinbach**[‡,6], **Thomas Moreau**[‡,7], **Fabio Muratore**[‡,8], **Julia Linhart**[‡,7], **Conor Durkan**[‡,9], **Julius Vetter**[1,2], **Benjamin Kurt Miller**[10], **Maternus Herold**[3,11,12], **Abolfazl Ziaeemehr**[13], **Matthijs Pals**[1,2], **Theo Gruner**[14], **Sebastian Bischoff**[1,2,15], **Anastasia N. Krouglova**[16,17], **Richard Gao**[1,2], **Janne K. Lappalainen**[1,2], **Bálint Mucsányi**[1,2,18], **Felix Pei**[19], **Auguste Schulz**[1,2], **Zinovia Stefanidi**[1,2], **Pedro L. C. Rodrigues**[20], **Cornelius Schröder**[1,2], **Faried Abu Zaid**[3], **Jonas Beck**[2,21], **Jaivardhan Kapoor**[1,2], **David S. Greenberg**[22,23], **Pedro J. Gonçalves**[17,24], **Jakob H. Macke**[1,2,25]

jan.boelts@mailbox.org, michael.deistler@uni-tuebingen.de, jakob.macke@uni-tuebingen.de

[1]Machine Learning in Science, University of Tübingen; [2]Tübingen AI Center; [3]TransferLab, appliedAI Institute for Europe; [4]ML Colab, Cluster ML in Science, University of Tübingen; [5]Google Research; [6]Helmholtz-Zentrum Dresden-Rossendorf; [7]Université Paris-Saclay, INRIA, CEA, Palaiseau, France; [8]Robert Bosch GmbH; [9]School of Informatics, University of Edinburgh; [10]University of Amsterdam; [11]Research and Innovation Center, BMW Group; [12]Institute for Applied Mathematics and Scientific Computing, University of the Bundeswehr Munich, Germany; [13]Aix Marseille, INSERM, INS, France; [14]TU Darmstadt, hessian.AI, Germany; [15]University Hospital Tübingen and M3 Research Center; [16]Faculty of Science, B-3000, KU Leuven, Belgium; [17]VIB-Neuroelectronics Research Flanders (NERF) and imec, Belgium; [18]Methods of Machine Learning, University of Tübingen; [19]Neuroscience Institute, Carnegie Mellon University; [20]Université Grenoble Alpes, INRIA, CNRS, Grenoble INP, LJK, France; [21]Hertie Institute for AI in Brain Health, University of Tübingen; [22]Institute of Coastal Systems - Analysis and Modeling; [23]Helmholtz AI; [24]Departments of Computer Science Electrical Engineering, KU Leuven, Belgium; [25]Department Empirical Inference, Max Planck Institute for Intelligent Systems, Tübingen, Germany

ABSTRACT

Scientists and engineers use simulators to model empirically observed phenomena. However, tuning the parameters of a simulator to ensure its outputs match observed data presents a significant challenge. Simulation-based inference (SBI) addresses this by enabling Bayesian inference for simulators, identifying parameters that match observed data and align with prior knowledge. Unlike traditional Bayesian inference, SBI only needs access to simulations from the model and does not require evaluations of the likelihood-function. In addition, SBI algorithms do not require gradients through the simulator, allow for massive parallelization of simulations, and can perform inference for different observations without further simulations or training, thereby amortizing inference. Over the past years, we have developed, maintained, and extended `sbi`, a PyTorch-based package[4] that implements Bayesian SBI algorithms based on neural networks. The `sbi` toolkit implements a wide range of inference methods, neural network architectures, sampling methods, and diagnostic tools. In addition, it provides well-tested default settings but also offers flexibility to fully customize every step of the simulation-based inference workflow. Taken together, the `sbi` toolkit enables scientists and engineers to apply state-of-the-art SBI methods to black-box simulators, opening up new possibilities for aligning simulations with empirically observed data.

1 Statement of need

Bayesian inference is a principled approach for determining parameters consistent with empirical observations: Given a prior over parameters, a forward-model (defining the likelihood), and observations, it returns a posterior distribution. The posterior distribution captures the entire space of parameters that are compatible with the observations and the prior

[*]Maintainer.
[†]Core contributor.
[‡]Major contributor.
[4]`sbi` is available at github.com/sbi-dev/sbi under the Apache 2.0 license.

and it quantifies parameter uncertainty. When the forward-model is given by a stochastic simulator, Bayesian inference can be challenging: (1) the forward-model can be slow to evaluate, making algorithms that rely on sequential evaluations of the likelihood (such as Markov-Chain Monte-Carlo, MCMC) impractical, (2) the simulator can be non-differentiable, prohibiting the use of gradient-based MCMC or variational inference (VI) methods, and (3) likelihood-evaluations can be intractable, meaning that we can only generate samples from the model, but not evaluate their likelihoods.

Recently, simulation-based inference (SBI) algorithms based on neural networks have been developed to overcome these limitations [1–3]. Unlike classical methods from Approximate Bayesian Computation (ABC [4]), these methods use neural networks to learn the relationship between parameters and simulation outputs. Neural SBI algorithms (1) allow for massive parallelization of simulations (in contrast with sequential evaluations in MCMC methods) (2) do not require gradients through the simulator, and (3) do not require evaluations of the likelihood but only samples from the simulator. Finally, many of these algorithms allow *amortized* inference, that is, after a large upfront cost of simulating data for the training phase, they can return the posterior distribution for any observation without requiring any further simulations or retraining.

To aid in the effective application of these algorithms to a wide range of problems, we developed the `sbi` toolkit. `sbi` implements a variety of state-of-the-art SBI algorithms, offering both high-level interfaces, extensive documentation and tutorials for practitioners, as well as low-level interfaces for experienced users and SBI researchers (giving full control over simulations, the training loop, and the sampling procedure). Since the original release of the `sbi` package [5], the community of contributors has expanded significantly, resulting in a large number of improvements that have made `sbi` more flexible, performant, and reliable. `sbi` now supports a wider range of amortized and sequential inference methods, neural network architectures (including normalizing flows, flow- and score-matching, and various embedding network architectures), samplers (including MCMC, variational inference, importance sampling, and rejection sampling), diagnostic tools, visualization tools, and a comprehensive set of tutorials on how to use these features.

The `sbi` package is already used extensively by the machine learning research community [6–18] but has also fostered the application of SBI in various fields of research [19–37].

2 Description

`sbi` is a flexible and extensive toolkit for running simulation-based Bayesian inference workflows. `sbi` supports any kind of (offline) simulator and prior, a wide range of inference methods, neural networks, and samplers, as well as diagnostic methods and analysis tools (Fig. 1).

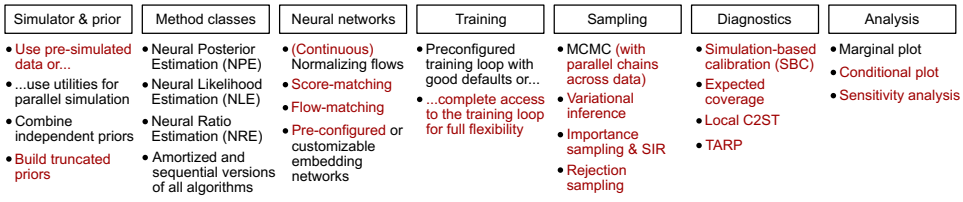


Figure 1: **Features of the `sbi` package.** Components that were added since the initial release described in Tejero-Cantero et al. [5] are marked in red.

A significant challenge in making SBI algorithms accessible to a broader community lies in accommodating diverse and complex simulators, as well as varying degrees of flexibility in each step of the inference process. To address this, `sbi` provides pre-configured defaults for all inference methods, but also allows full customization of every step in the process (including simulation, training, sampling, diagnostics and analysis).

Simulator & prior: The `sbi` toolkit requires only simulation parameters and simulated data as input, and no direct access to the simulator itself. However, if the simulator can be provided as a Python callable, `sbi` can optionally parallelize running the simulations from a given prior using Joblib [38]. Additionally, `sbi` can automatically handle failed simulations or missing values, it supports both discrete and continuous parameters and observations (or mixtures thereof) and it provides utilities to flexibly define priors.

Methods: `sbi` implements a wide range of neural network-based SBI algorithms, among them Neural Posterior Estimation (NPE) with various conditional estimators, Neural Likelihood Estimation (NLE), and Neural Ratio Estimation (NRE). Each of these methods can be run either in an *amortized* mode, where the neural network is trained once on a set of pre-existing simulations results and then performs inference on *any* observation without further simulations or retraining, or in a *sequential* mode, where inference is focused on one observation to improve simulation efficiency with active learning.

Neural networks and training: `sbi` implements a wide variety of state-of-the-art conditional density estimators for NPE and NLE, including normalizing flows [39, 40] (via nflows and Zuko [41, 42]), diffusion models [43–45], mixture

density networks [46], and flow matching [47, 48] (via Zuko [42]), as well as ensembles of any of these networks. sbi also implements a large set of embedding networks that can automatically learn summary statistics of (potentially) high-dimensional simulation outputs (including multi-layer-perceptrons, convolutional networks, and permutation invariant networks). The neural networks can be trained with a pre-configured training loop with established default values, but sbi also allows full access over the training loop when desired.

Sampling: For NLE and NRE, sbi implements a large range of samplers, including MCMC (with chains vectorized across observations), variational inference, rejection sampling, or importance sampling, as well as wrappers to use MCMC samplers from Pyro and PyMC [49, 50]. sbi can perform inference for single observations or for multiple i.i.d. observations, and can use importance sampling to correct for potential inaccuracies in the posterior if the likelihood is available.

Diagnostics and analysis: The sbi toolkit also implements a large set of diagnostic tools, such as simulation-based calibration (SBC) [51], expected coverage [6, 16], local C2ST [17], and TARP [52]. Additionally, sbi offers visualization tools, including marginal and conditional corner plots to visualize high-dimensional distributions, calibration plots, and wrappers for Arviz [53] diagnostic plots.

With sbi, our goal is to advance scientific discovery and computational engineering by making Bayesian inference accessible to a broad range of models, including those with inaccessible likelihoods, and to a broader range of users, including both machine learning researchers and domain-practitioners. We have created an open architecture and embraced community-driven development practices to encourage collaboration with other machine learning researchers and applied scientists to join us in this long-term vision.

3 Related software

Since the original release of the sbi package, several other packages that implement neural network-based SBI algorithms have emerged. The Lampe [54] package offers neural posterior and neural ratio estimation, primarily targeting SBI researchers with a low-level API and full flexibility over the training loop[5]. The BayesFlow [55] package focuses on a set of amortized SBI algorithms based on posterior and likelihood estimation. The Swyft [56] package specializes in algorithms based on neural ratio estimation. The sbijax [57] package implements a set of inference methods in JAX.

Acknowledgements

This work has been supported by the German Federal Ministry of Education and Research (BMBF, projects 'Simalesam', FKZ 01IS21055 A-B and 'DeepHumanVision', FKZ: 031L0197B, and the Tübingen AI Center FKZ: 01IS18039A), the German Research Foundation (DFG) through Germany's Excellence Strategy (EXC-Number 2064/1, PN 390727645) and SFB1233 (PN 276693517), SFB 1089 (PN 227953431), SPP 2041 (PN 34721065), SPP 2041 'Computational Connectomics', SPP 2298-2 (PN 543917411), SFB 1233 'Robust Vision', and Germany's Excellence Strategy EXC-Number 2064/1/Project number 390727645, the 'Certification and Foundations of Safe Machine Learning Systems in Healthcare' project funded by the Carl Zeiss Foundation, the Else Kröner Fresenius Stiftung (Project ClinbrAIn), and the European Union (ERC, "DeepCoMechTome", ref. 101089288). CD was supported by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh. BKM is part of the ELLIS PhD program, receiving travel support from the ELISE mobility program which has received funding from the European Union's Horizon 2020 research and innovation programme under ELISE grant agreement No 951847. DSG is supported by Helmholtz AI. JL is a recipient of the Pierre-Aguilar Scholarship and thankful for the funding of the Capital Fund Management (CFM). ANK is supported by an FWO grant (G097022N). TG was supported by "Third Wave of AI", funded by the Excellence Program of the Hessian Ministry of Higher Education, Science, Research and Art. TM and PLCR were supported from a national grant managed by the French National Research Agency (Agence Nationale de la Recherche) attributed to the ExaDoST project of the NumPEx PEPR program, under the reference ANR-22-EXNU-0004.. PS is supported by the Helmholtz Association Initiative and Networking Fund through the Helmholtz AI platform grant. MD, MG, GM, JV, MP, SB, JKL, AS, ZS, JB are members of the International Max Planck Research School for Intelligent Systems (IMPRS-IS).

References

[1] George Papamakarios and Iain Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. *Advances in neural information processing systems*, 29, 2016.

[2] George Papamakarios, David Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 837–848. PMLR, 2019.

[5] Lampe stopped being maintained in July 2024.

[3] Joeri Hermans, Volodimir Begy, and Gilles Louppe. Likelihood-free mcmc with amortized approximate ratio estimators. In *International Conference on Machine Learning*, pages 4239–4248. PMLR, 2020.

[4] S. A. Sisson, Fan Y., and Beaumont M. A. Overview of abc. In *Handbook of Approximate Bayesian Computation*, Chapman & Hall/CRC Handbooks of Modern Statistical Methods, chapter 1. CRC Press, Taylor & Francis Group, 2018. ISBN 9781439881507. doi: 10.1201/9781315117195.

[5] Alvaro Tejero-Cantero, Jan Boelts, Michael Deistler, Jan-Matthis Lueckmann, Conor Durkan, Pedro J. Goncalves, David S. Greenberg, and Jakob H. Macke. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52):2505, 2020.

[6] Michael Deistler, Pedro J. Goncalves, and Jakob H. Macke. Truncated proposals for scalable and hassle-free simulation-based inference. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[7] Manuel Glöckler, Michael Deistler, and Jakob H Macke. Variational methods for simulation-based inference. In *International Conference on Learning Representations*, 2022.

[8] Fabio Muratore, Theo Gruner, Florian Wiese, Boris Belousov, Michael Gienger, and Jan Peters. Neural posterior domain randomization. In *Conference on Robot Learning*, pages 1532–1542. PMLR, 2022.

[9] Manuel Gloeckler, Michael Deistler, and Jakob H Macke. Adversarial robustness of amortized bayesian inference. In *International Conference on Machine Learning*, pages 11493–11524. PMLR, 2023.

[10] Joel Dyer, Patrick Cannon, J. Doyne Farmer, and Sebastian M Schmon. Calibrating agent-based models to microdata with graph neural networks. In *ICML 2022 Workshop AI for Agent-Based Modelling*, 2022.

[11] Samuel Wiqvist, Jes Frellsen, and Umberto Picchini. Sequential neural posterior and likelihood approximation. *arXiv preprint arXiv:2102.06522*, 2021.

[12] A Spurio Mancini, MM Docherty, MA Price, and JD McEwen. Bayesian model comparison for simulation-based inference. *RAS Techniques and Instruments*, 2(1):710–722, 2023.

[13] Simon Dirmeier, Carlo Albert, and Fernando Perez-Cruz. Simulation-based inference using surjective sequential neural likelihood estimation. *arXiv preprint arXiv:2308.01054*, 2023.

[14] Richard Gao, Michael Deistler, and Jakob H Macke. Generalized bayesian inference for scientific simulators via amortized cost estimation. *Advances in Neural Information Processing Systems*, 36:80191–80219, 2023.

[15] Manuel Gloeckler, Michael Deistler, Christian Dietrich Weilbach, Frank Wood, and Jakob H. Macke. All-in-one simulation-based inference. In *Forty-first International Conference on Machine Learning*, 2024.

[16] Joeri Hermans, Arnaud Delaunoy, François Rozet, Antoine Wehenkel, and Gilles Louppe. A crisis in simulation-based inference? beware, your posterior approximations can be unfaithful. *Transactions on Machine Learning Research*, 2022.

[17] Julia Linhart, Alexandre Gramfort, and Pedro Rodrigues. L-c2st: Local diagnostics for posterior approximations in simulation-based inference. *Advances in Neural Information Processing Systems*, 36, 2024.

[18] Jan Boelts, Jan-Matthis Lueckmann, Richard Gao, and Jakob H Macke. Flexible and efficient simulation-based inference for models of decision-making. *Elife*, 11:e77220, 2022.

[19] Lukas N Groschner, Jonatan G Malis, Birte Zuidinga, and Alexander Borst. A biophysical account of multiplication by a single neuron. *Nature*, 603(7899):119–123, 2022.

[20] Vladyslav Bondarenko, Mikhail Nikolaev, Dimitri Kromm, Roman Belousov, Adrian Wolny, Marloes Blotenburg, Peter Zeller, Saba Rezakhani, Johannes Hugger, Virginie Uhlmann, et al. Embryo-uterine interaction coordinates mouse embryogenesis during implantation. *The EMBO Journal*, 42(17):e113280, 2023.

[21] Basile Confavreux, Poornima Ramesh, Pedro J Goncalves, Jakob H Macke, and Tim Vogels. Meta-learning families of plasticity rules in recurrent spiking networks using simulation-based inference. *Advances in Neural Information Processing Systems*, 36:13545–13558, 2023.

[22] Dylan Myers-Joseph, Katharina A Wilmes, Marian Fernandez-Otero, Claudia Clopath, and Adil G Khan. Disinhibition by vip interneurons is orthogonal to cross-modal attentional modulation in primary visual cortex. *Neuron*, 112(4):628–645, 2024.

[23] Grace Avecilla, Julie N Chuong, Fangfei Li, Gavin Sherlock, David Gresham, and Yoav Ram. Neural networks enable efficient and accurate simulation-based inference of evolutionary parameters from adaptation dynamics. *PLoS biology*, 20(5):e3001633, 2022.

[24] Eric Lowet, Daniel J Sheehan, Ulises Chialva, Rodrigo De Oliveira Pena, Rebecca A Mount, Sheng Xiao, Samuel L Zhou, Hua-an Tseng, Howard Gritton, Sanaya Shroff, et al. Theta and gamma rhythmic coding through two spike output modes in the hippocampus during spatial navigation. *Cell reports*, 42(8), 2023.

[25] Yves Bernaerts, Michael Deistler, Pedro J Gonçalves, Jonas Beck, Marcel Stimberg, Federico Scala, Andreas S Tolias, Jakob Macke, Dmitry Kobak, and Philipp Berens. Combined statistical-mechanistic modeling links ion channel genes to physiology of cortical neuron types. *bioRxiv*, pages 2023–03, 2023.

[26] Siddharth Mishra-Sharma and Kyle Cranmer. Neural simulation-based inference approach for characterizing the galactic center γ-ray excess. *Physical Review D*, 105(6):063017, 2022.

[27] Joel Dyer, Patrick Cannon, J Doyne Farmer, and Sebastian Schmon. Black-box bayesian inference for economic agent-based models. *arXiv preprint arXiv:2202.00625*, 2022.

[28] Meysam Hashemi, Anirudh N Vattikonda, Jayant Jha, Viktor Sip, Marmaduke M Woodman, Fabrice Bartolomei, and Viktor K Jirsa. Amortized bayesian inference on generative dynamical network models of epilepsy using deep neural density estimators. *Neural Networks*, 163:178–194, 2023.

[29] ChangHoon Hahn and Peter Melchior. Accelerated bayesian sed modeling using amortized neural posterior estimation. *The Astrophysical Journal*, 938(1):11, 2022.

[30] Pablo Lemos, Liam Parker, ChangHoon Hahn, Shirley Ho, Michael Eickenberg, Jiamin Hou, Elena Massara, Chirag Modi, Azadeh Moradinezhad Dizgah, Bruno Régaldo-Saint Blancard, et al. Field-level simulation-based inference of galaxy clustering with convolutional neural networks. *Physical Review D*, 109(8):083536, 2024.

[31] Michael Deistler, Jakob H Macke, and Pedro J Gonçalves. Energy-efficient network activity from disparate circuit parameters. *Proceedings of the National Academy of Sciences*, 119(44):e2207632119, 2022.

[32] Nina Rößler, Tassilo Jungenitz, Albrecht Sigler, Alexander Bird, Martin Mittag, Jeong Seop Rhee, Thomas Deller, Hermann Cuntz, Nils Brose, Stephan W Schwarzacher, et al. Skewed distribution of spines is independent of presynaptic transmitter release and synaptic plasticity, and emerges early during adult neurogenesis. *Open Biology*, 13(8):230063, 2023.

[33] Lars Dingeldein, Pilar Cossio, and Roberto Covino. Simulation-based inference of single-molecule force spectroscopy. *Machine Learning: Science and Technology*, 4(2):025009, 2023.

[34] Huaqing Jin, Parul Verma, Fei Jiang, Srikantan S Nagarajan, and Ashish Raj. Bayesian inference of a spectral graph model for brain oscillations. *NeuroImage*, 279:120278, 2023.

[35] Jan Boelts, Philipp Harth, Richard Gao, Daniel Udvary, Felipe Yáñez, Daniel Baum, Hans-Christian Hege, Marcel Oberlaender, and Jakob H Macke. Simulation-based inference for efficient identification of generative models in computational connectomics. *PLOS Computational Biology*, 19(9):e1011406, 2023.

[36] Richard Gao, Michael Deistler, Auguste Schulz, Pedro J Gonçalves, and Jakob H Macke. Deep inverse modeling reveals dynamic-dependent invariances in neural circuit mechanisms. *bioRxiv*, pages 2024–08, 2024.

[37] Xiaoyu Wang, Ryan P. Kelly, Adrianne L. Jenner, David J. Warne, and Christopher Drovandi. A comprehensive guide to simulation-based inference in computational biology, 2024.

[38] Gael Varoquaux. joblib. https://github.com/joblib/joblib, 2008.

[39] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

[40] David Greenberg, Marcel Nonnenmacher, and Jakob Macke. Automatic posterior transformation for likelihood-free inference. In *International Conference on Machine Learning*, pages 2404–2414. PMLR, 2019.

[41] Conor Durkan, Artur Bekasov, George Papamakarios, and Iain Murray. nflows: Normalizing flows in pytorch. https://github.com/bayesiains/nflows, 2019.

[42] Francois Rozet. Zuko - normalizing flows in pytorch. https://github.com/probabilists/zuko, 2023.

[43] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[44] Tomas Geffner, George Papamakarios, and Andriy Mnih. Compositional score modeling for simulation-based inference. In *International Conference on Machine Learning*, pages 11098–11116. PMLR, 2023.

[45] Jack Simons, Louis Sharrock, Song Liu, and Mark Beaumont. Neural score estimation: Likelihood-free inference with conditional score based diffusion models. In *Fifth Symposium on Advances in Approximate Bayesian Inference*, 2023.

[46] C M Bishop. Mixture density networks. *Technical Report. Aston University, Birmingham*, 1994.

[47] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

[48] Jonas Bernhard Wildberger, Maximilian Dax, Simon Buchholz, Stephen R Green, Jakob H. Macke, and Bernhard Schölkopf. Flow matching for scalable simulation-based inference. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[49] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019.

[50] Oriol Abril-Pla, Virgile Andreani, Colin Carroll, Larry Dong, Christopher J Fonnesbeck, Maxim Kochurov, Ravin Kumar, Junpeng Lao, Christian C Luhmann, Osvaldo A Martin, et al. Pymc: a modern, and comprehensive probabilistic programming framework in python. *PeerJ Computer Science*, 9:e1516, 2023.

[51] Sean Talts, Michael Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. Validating bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:1804.06788*, 2018.

[52] Pablo Lemos, Adam Coogan, Yashar Hezaveh, and Laurence Perreault-Levasseur. Sampling-based accuracy testing of posterior estimators for general inference. In *International Conference on Machine Learning*, pages 19256–19273. PMLR, 2023.

[53] Ravin Kumar, Colin Carroll, Ari Hartikainen, and Osvaldo Martin. Arviz a unified library for exploratory analysis of bayesian models in python. *Journal of Open Source Software*, 4(33):1143, 2019.

[54] Probabilists. Lampe: Likelihood-free amortized posterior estimation with pytorch. https://github.com/probabilists/lampe, 2024. Version 0.9.0.

[55] Stefan T. Radev, Marvin Schmitt, Lukas Schumacher, Lasse Elsemüller, Valentin Pratz, Yannik Schälte, Ullrich Köthe, and Paul-Christian Bürkner. BayesFlow: Amortized Bayesian workflows with neural networks. *Journal of Open Source Software*, 8(89):5702, 2023.

[56] undark lab. Swyft: A system for scientific simulation-based inference at scale. https://github.com/undark-lab/swyft, 2023. Version 0.4.5.

[57] Simon Dirmeier, Simone Ulzega, Antonietta Mira, and Carlo Albert. Simulation-based inference with the python package sbijax, 2024.

Differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics

Michael Deistler[1,2], **Kyra L. Kadhim**[2,3], **Matthijs Pals**[1,2], **Jonas Beck**[2,3], **Ziwei Huang**[2,3], **Manuel Gloeckler**[1,2], **Janne K. Lappalainen**[1,2], **Cornelius Schröder**[1,2], **Philipp Berens**[2,3], **Pedro J. Gonçalves**[1,2,4,5], **Jakob H. Macke**[1,2,6]

[1]Machine Learning in Science, University of Tübingen; [2]Tübingen AI Center, Tübingen, Germany; [3]Hertie Institute for AI in Brain Health, University of Tübingen, Tübingen, Germany; [4]VIB-Neuroelectronics Research Flanders (NERF); [5]imec, Belgium; [6]Department Empirical Inference, Max Planck Institute for Intelligent Systems, Tübingen, Germany

Abstract Biophysical neuron models provide insights into cellular mechanisms underlying neural computations. However, a central challenge has been the question of how to identify the parameters of detailed biophysical models such that they match physiological measurements at scale or such that they perform computational tasks. Here, we describe a framework for simulation of detailed biophysical models in neuroscience—JAXLEY—which addresses this challenge. By making use of automatic differentiation and GPU acceleration, JAXLEY opens up the possibility to efficiently optimize large-scale biophysical models with gradient descent. We show that JAXLEY can learn parameters of biophysical neuron models with several hundreds of parameters to match voltage or two photon calcium recordings, sometimes orders of magnitude more efficiently than previous methods. We then demonstrate that JAXLEY makes it possible to train biophysical neuron models to perform computational tasks. We train a recurrent neural network to perform working memory tasks, and a feedforward network of morphologically detailed neurons with 100,000 parameters to solve a computer vision task. Our analyses show that JAXLEY dramatically improves the ability to build large-scale data- or task-constrained biophysical models, creating unprecedented opportunities for investigating the mechanisms underlying neural computations across multiple scales.

Introduction

Computational models are used to devise hypotheses about neural systems and to design experiments to investigate them. When building such models, a central question is how much detail they should include: Models of neural systems range from simple rate-based point neuron models to morphologically detailed biophysical neuron models [1–4]. The latter provide fine-grained mechanistic explanations of cellular processes underlying neural activity, typically described as systems of ordinary differential equations [5–10].

However, it has been highly challenging for neuroscientists to create biophysical models that can explain physiological measurements [10–13] or that can perform computational tasks [14–16]. It is hardly ever possible to directly measure all relevant microscopic properties of the system with sufficient precision to constrain all parameters directly, necessitating the use of *inference* or fitting-approaches to optimize free model parameters [17]. However, finding the right parameters for even a single neuron model with only a few parameters can be difficult [13, 18], and large-scale morphologically detailed biophysical network models may have thousands of free parameters. Therefore, neuroscientists work with simplified models that sacrifice biophysical detail for interpretability and computational efficiency (e.g.,

For correspondence: michael.deistler@uni-tuebingen.de; jakob.macke@uni-tuebingen.de

rate-based or leaky-integrate and fire neuron models, or models with simplified morphologies) [17, 19–21], or build biophysical simulations in a purely bottom-up way [11, 22, 23], without constraining the resulting network models by data or computational tasks as a whole.

Recently, in many domains of science such as particle physics, geoscience, and quantum chemistry, *differentiable*, GPU-accelerated simulators have enabled parameter inference for even complicated models using modern automatic differentiation techniques [24–28]. Such differentiable simulators make it possible to train simulators with gradient-descent methods from deep learning [29]: Backpropagation of error ('backprop') makes the computational cost of computing the gradient of the model with respect to the parameters independent of the number of parameters, making it possible to efficiently fit large models. In addition, GPU acceleration allows computing the gradient for many inputs (or model configurations) in parallel, which allows fitting simulations to large datasets with stochastic gradient descent [30].

Numerical solvers for biophysical models in neuroscience are used extensively, and several software packages exist [31, 32], in particular the commonly used NEURON simulation environment [33–35]. Yet, none of these simulators allows performing backprop, and currently used simulation engines are primarily CPU-based, with GPU-functionality only added post-hoc [36–38]. As a consequence, state-of-the-art methods for parameter estimation in biophysical neuron models are based on gradient-free approaches such as genetic algorithms [18, 39] or simulation-based inference [40], which do not scale to models with many parameters.

In principle, biophysical models of neurons are mathematically differentiable with respect to the parameters [41]. Thus, an efficient and scalable differentiable simulator would open up the possibility of optimizing such models with gradient descent. Such simulators would also need to handle the fact that the computation graph—a crucial ingredient for backpropagation of error—may be too large to keep in memory [42].

To close this gap, we developed a new framework for simulation and inference of (morphologically detailed) biophysical models in neuroscience, called JAXLEY. JAXLEY has in-built automatic differentiation capabilities and makes training efficient with native GPU-acceleration and just-in-time (JIT) compilation, building on computationally efficient engineering solutions developed in the machine learning community. This framework allows researchers to simulate large-scale biophysical models with thousands of parameters efficiently and even fit such models either to physiological data or to perform a computational task.

We demonstrate the power of this approach on a series of tasks covering different scales, data modalities, and levels of biophysical detail. Regarding fitting physiological data, we first show that we can efficiently recover the parameters of multicompartment models of single neurons based on experimental intracellular recordings or simulated voltage imaging recordings using gradient descent. In some cases, gradient descent is orders of magnitude more accurate and efficient than previous gradient-free methods. Second, we show that we can fit synaptic and cellular parameters of a hybrid model of the presynaptic circuit of a retinal ganglion cell to match dendritic calcium recordings in response to light stimuli. Regarding solving computational tasks, we show that we can train recurrent networks of Hodgkin–Huxley-type models (with detailed channel dynamics and multiple compartments) to solve working memory tasks. Finally, we build a large feedforward network with more than 800 cells, of which 64 are modelled at full morphological detail, and show that such a network can solve the classical MNIST task from machine learning without any additional nonlinearities. Our numerical experiments show that JAXLEY is a flexible and easy-to-use simulation framework, running natively on CPUs, GPUs or TPUs and which, unlike previous neuroscience simulators, makes efficient automatic differentiation possible, unlocking new possibilities for data-driven biophysical simulations in neuroscience.

Results

JAXLEY: A new toolbox for simulation and inference in neuroscience

Our goal was to optimize biophysical neuron models so that they either quantitatively fit data (e.g., match experimental recordings such as voltage or calcium measurements [11, 12, 18, 44]) or can achieve high performance in computational tasks such as short-term memory retrieval [45–47] or image classification [16, 48] (Fig. 1a). As the datasets that define

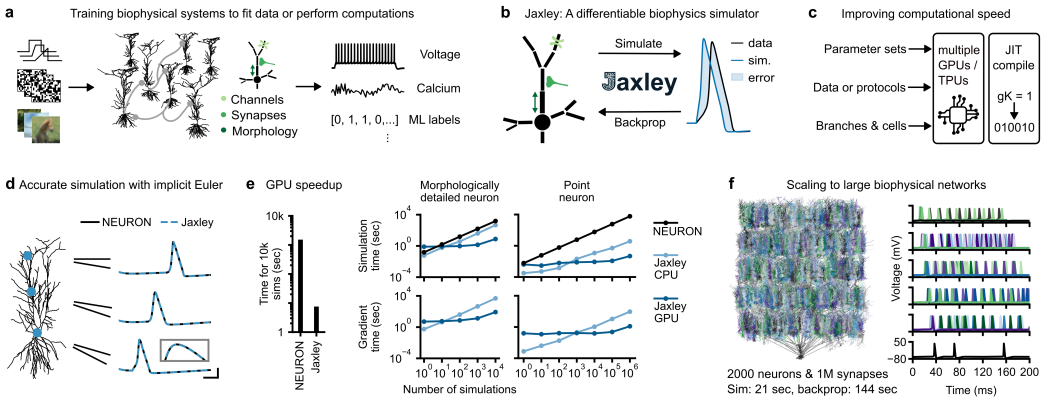


Figure 1. Differentiable simulation enables training biophysical neuron models. (a) Schematic of goal. Given input stimuli (left, step currents, or sensory stimuli encoded into input currents), we aim to train biophysically detailed neural networks (middle) either to match physiological recordings (right, voltage or calcium recordings), or to solve supervised machine learning tasks by predicting labels. To perform such tasks, we have to infer biophysical parameters such as channel conductances, synaptic conductances, or morphological parameters such as branch radii or lengths. (b) Schematic of method. Our simulator, JAXLEY, allows to simulate biophysically detailed neural systems, and it can also compute the gradient of any cost function with respect to biophysical parameters via backpropagation of error. (c) Computational efficiency of JAXLEY. JAXLEY can parallelize simulations on (multiple) GPUs/TPUs across parameter sets, data (e.g., protocols), or branches and cells in a network. It can also just-in-time (JIT) compile code to further speed up simulation and training. (d) Simulated voltage traces at three locations, based on a reconstruction of a CA1 neuron [43] in response to a step current obtained with the NEURON simulator and with JAXLEY. Inset is a zoom-in to the peak of the action potential. Scalebars: 3 ms and 30 mV. (e) Left: Time to run 10k simulations with NEURON on CPU and with JAXLEY on GPU. Right: Simulation time (top) for the CA1 neuron shown in panel d and for a point neuron, as a function of number of simulations. Bottom: Same as top, for computing the gradient with backprop. (f) JAXLEY scales to large networks. JAXLEY can simulate and differentiate a biophysically detailed network built from reconstructions of CA1 neurons (left, network consists of 2,000 neurons and 1M synapses) in response to step currents to the first layer (voltage responses to the right). Runtimes were evaluated on an A100 GPU.

these tasks are becoming increasingly complex, one has to adjust many (potentially thousands of) free parameters governing the behavior of ion channels (e.g., maximal conductance), synapses (e.g., synaptic conductance or time constant), or neural morphologies (e.g., radius or branch length). Inspired by the capabilities of deep learning to adjust millions (or even billions) of parameters given large datasets, we here suggest to adjust these parameters with gradient descent and to speed up training with GPUs.

No current toolbox for biophysical simulation, however, allows to perform backprop as required for efficient gradient descent. Therefore, we built JAXLEY, a new Python toolbox for simulation of biophysical neuron models. JAXLEY allows to compute the gradient with respect to biophysical parameters with backprop, provides utilities to robustly perform gradient descent, and speeds up simulation and training using GPU acceleration and just-in-time (JIT) compilation (Fig. 1b). To achieve this, JAXLEY implements numerical routines required for efficiently simulating biophysically-detailed neural systems, so-called implicit Euler solvers, in the deep learning Python framework JAX [49]. In particular, for multicompartment models, bespoke implicit solvers are key to accurate and stable simulation [50], but are not provided by any other toolbox for simulation of differential equations [51, 52]. Building on JAX, JAXLEY inherits the capability to perform automatic differentiation, such that errors can be propagated back through the implicit Euler solvers, which makes it possible obtain the gradient with respect to any parameter.

Training biophysical models with gradient descent leads to instabilities resulting from parameters having different

scales, networks having a large computation graph [42], loss surfaces being non-convex [53], and so on. JAXLEY implements methods that have been developed to overcome these specific issues in deep neural networks (Supp. Fig. S1). JAXLEY also provides native support for parallelization: It implements the differential equations such that, for example, different branches of a single neural morphology can be solved in parallel on GPUs [36, 38, 54]. This results in large speed ups for cell and network simulation and differentiation. In addition, JAXLEY can parallelize across stimuli or parameter sets, providing large speed-ups for datasets (via stimulus parallelization) or for parameter sweeps (via parameter parallelization, Fig. 1c). Finally, JAXLEY just-in-time (JIT) compiles code, making it (at least) as fast as previous simulators which are written in compiled programming languages.

Taken together, the abilities of JAXLEY to compute the gradient with backprop, to perform robust optimization with gradient descent, and to parallelize simulations across stimuli and cells on GPUs open up the possibility to train large biophysical networks with thousands of parameters. Furthermore, we designed JAXLEY with a user-friendly interface, allowing neuroscientists to build biophysical models (e.g., for inserting recordings, stimuli, and channels into various branches or cells) and to use automatic differentiation and GPU parallelization. In a dedicated library open to the community, it also implements different types of connectivity structures (e.g., dense or sparse connectivity), a growing set of ion channel models, and utilities that are required to perform robust training with gradient descent (e.g., parameter transformations, multi-level checkpointing [55], specific optimizers for non-convex loss surfaces [56]). JAXLEY is fully written in Python, which will make it easy for the community to use and to add functionality to it[1].

JAXLEY is accurate, fast, and scalable

We benchmarked the accuracy, speed, and scalability of JAXLEY for simulation of biophysical models. First, we evaluated the accuracy of JAXLEY and created biophysically-detailed multicompartment models of a CA1 pyramidal cell from rat hippocampus [43, 57] and of four layer 5 neurons from the mouse visual area from the Allen Cell Types Database [58]. Every model contained sodium, potassium, and leak channels in all branches. We stimulated the soma and recorded the voltage at three locations across the dendritic tree. JAXLEY matched the voltages of the NEURON simulator at sub-microsecond and sub-millivolt resolution (Fig. 1d). Across all five cells, input currents with ten different amplitudes, and three recording sites, the deviation of spike time between the NEURON simulator and JAXLEY was at most 0.05 ms and the difference in spike amplitude was at most 0.02 mV (Supp. Fig. S2).

Next, we evaluated the simulation speed of JAXLEY on CPU and GPU. We simulated the above described CA1 cell, as well as a single compartment model, for 20 ms. To demonstrate the parallelization capabilities of JAXLEY, we also evaluated the runtimes for running multiple simulations in parallel with different parameter sets. On GPU, JAXLEY was much faster for large systems or many simulations, with a speed up of around two orders of magnitude (Fig. 1e). For single compartment neurons, JAXLEY could parallelize the simulation of up to 1 million neurons, thereby allowing fast parameter sweeps. On CPU, JAXLEY was similar to NEURON in speed for multicompartment models and was faster than NEURON for single compartment models because JAXLEY can vectorize code, which avoids slow for-loops across parameter sets.

We then evaluated the computational cost of computing the gradient with JAXLEY (Fig. 1e, bottom). For backpropagation, the forward pass must be stored in-memory, which can easily correspond to terabytes of data for large neural systems. To overcome this, JAXLEY implements multi-level checkpointing [55], which reduces memory usage by strategically saving and recomputing intermediate states of the system of differential equations. Overall, we found that computing the gradient was slightly more expensive than the simulation itself: Depending on the simulation device (CPU/GPU/TPU), the number of simulations, the number of branches and channels, the simulated time, and the loss function, computing the gradient was between 3 and 20 times more expensive than running the simulation itself (Fig. 1f, bottom) [59].

Finally, we show that in addition to parallelizing across parameters (or across stimuli), JAXLEY can parallelize across

[1] JAXLEY is openly available at https://github.com/jaxleyverse/jaxley

branches or compartments in a network, allowing to simulate and differentiate large networks of biophysical neuron models. To demonstrate this, we built a multi-layer neural network consisting of 2,000 morphologically detailed neurons with Hodgkin–Huxley dynamics, connected by 1 million biophysical synapses (3.92 million differential equation states in total, Fig. 1f). On a single A100 GPU, JAXLEY computed 200 ms (i.e., 8,000 steps at $\Delta t = 0.025$ ms) of simulated time in 21 seconds. We then used backprop to compute the gradient with respect to all membrane and synaptic conductances in this network (3.2 million parameters in total), which took 144 seconds. Estimating the gradient with finite differences—as would be required for packages that do not support backprop—would take more than two years (3.2 million forward passes, 21 seconds each).

Fitting biophysical single neuron models to intracellular recordings

Having demonstrated the accuracy and speed of JAXLEY, we applied it to a series of tasks which demonstrate how JAXLEY opens up new opportunities for building task- or data-driven biophysically-detailed neuroscience models in a range of scenarios. We show that JAXLEY can fit biophysical models to physiological measurements such as voltage and calcium recordings, sometimes vastly more accurately and efficiently than gradient-free methods, and we demonstrate that JAXLEY allows to fit large-scale biophysical networks with up to 100k parameters to computational tasks such as memory retrieval or image recognition.

As a first proof-of-principle, we applied JAXLEY to fit single neuron models with few parameters. As we will show, even in these models in which gradient-free methods such as genetic algorithms excel and are used extensively, gradient descent can be competitive and sometimes even outperform state-of-the-art genetic algorithms. We built a biophysical neuron model based on a reconstruction of a layer 5 pyramidal cell (L5PC) (Fig. 2a). The model had nine different channels in the apical and basal dendrite, the soma, and the axon [39], with a total of 19 free parameters, including maximal channel conductances and dynamics of the calcium pumps. We learned these parameters from a synthetic somatic voltage recording given a somatic step current stimulus with a known set of ground-truth parameters (Fig. 2b, top).

We used gradient descent to identify parameter sets which minimize the mean absolute error to summary statistics of the voltage trace. Since gradient descent requires differentiable summary statistics, but commonly used summary statistics of intracellular recordings—such as spike count—can be discrete or non-differentiable, we used mean and standard deviation of the voltage in two time windows [40]. Starting from randomly initialized parameters, gradient descent required only nine steps (median across ten runs) to find models whose voltage traces are visually similar to the observation (Fig. 2b, bottom). A state-of-the-art indicator-based genetic algorithm (IBEA) required similarly many iterations, although each iteration of the genetic algorithm used ten simulations. As a consequence, gradient descent required almost ten times fewer simulations than the genetic algorithm, and, despite the additional cost of backpropagation, found good parameter sets in less runtime than the genetic algorithm on CPU (Fig. 2c).

We then employed gradient descent to identify parameters that match patch-clamp recordings from four cells from the Allen Cell Types Database which had an axon initial segment reconstructed (because many of our parameters were axonal, Fig. 2d, top) [58]. We again defined windows of the voltage for summary statistics, inserted the same set of ion channels, and used the same set of free parameters (see Methods). Due to the length of the recordings (1 s vs 100 ms in the synthetic experiments), this was a much more challenging problem as it could lead to exploding or vanishing gradients. Despite this, gradient descent found parameter sets whose voltage traces closely resembled experimental recordings (Fig. 2d, bottom, additional fits in Supp. Fig. S3). For both models, gradient descent was more simulation efficient than the genetic algorithms and had similar runtime (Supp. Fig. S4). Overall, these results demonstrate the ability of gradient descent to fit biophysical models to intracellular recordings, being competitive with state-of-the-art genetic algorithms even on tasks for which those have been extensively optimized.

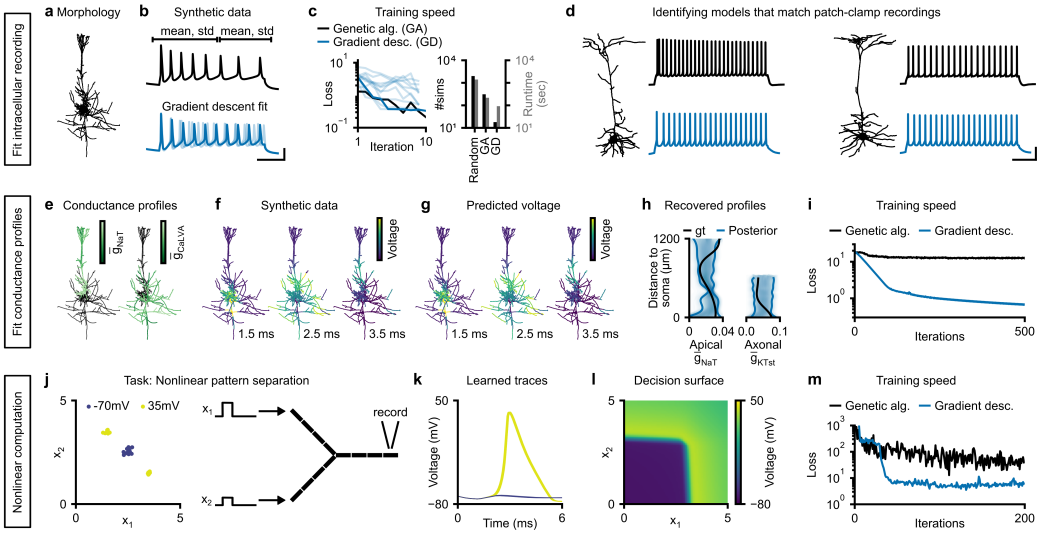


Figure 2. Inferring single-neuron models with gradient descent. Task 1 (a) We optimize 19 parameters of a biophysical neuron model which is based on the morphology of a layer 5 pyramidal cell (L5PC) morphology. (b) Top: Synthetic somatic voltage recording (black) and windows that are used to compute summary statistics (top). Bottom: Fits obtained with gradient descent. Best fit in dark blue, fits from five best independent runs in light blue. Scalebars: 20 ms and 30 mV. (c) Left: Loss value of individual gradient descent runs (light blue), their minimum (dark blue), in comparison to the minimum loss across ten genetic algorithm runs (black). Right: Average number of simulations and runtime required to find a set of parameters with loss smaller than 0.55 with random sampling, genetic algorithm, and gradient descent. (d) We also fit morphologically detailed models to patch-clamp recordings (black) in response to step currents from the Allen Cell Types Database. Gradient descent fit in blue. Additional models in Supp. Fig. S3. Scalebars: 200 ms and 30 mV. **Task 2** (e) We optimize conductance profiles of the same L5PC morphology, leading to 1390 parameters. Synthetic ground-truth conductance profiles vary as a function of distance to the soma. (f) Simulated voltages given the synthetic conductance profile after 1.5, 2.5, and 3.5 ms. (g) Predicted voltages of gradient descent fit closely match synthetic observation. (h) Ground truth conductance profile as a function of distance from the soma (black) and 90 % confidence interval obtained with multi-chain gradient-based Hamiltonian Monte-Carlo. (i) Loss as a function of the number of iterations for gradient descent and genetic algorithm. **Task 3** (j) We optimize parameters of a simplified morphology with twelve compartments (right) to solve a nonlinear pattern separation task (left). (k) Voltage traces of model found with gradient descent. (l) Decision surface of the model reveals nonlinear single-neuron computation. (m) Minimum loss across ten independent runs as a function of the number of iterations for gradient descent and genetic algorithm.

Fitting biophysical single neuron models with many parameters

How does gradient descent scale to models with large numbers of parameters? We demonstrate here that, in contrast to genetic algorithms, gradient descent allows to optimize a single neuron model with 1390 parameters.

We used the above described model of a L5PC with a diverse set of active conductances. Unlike in the above experiments, we fit the maximal conductance of ion channels in every branch in the morphology [60], thereby allowing to model effects of non-uniform conductance profiles [61, 62]. This increased the number of free parameters to 1390. To generate a synthetic recording, we assigned a different maximal conductance to each branch (sampled from a Gaussian process, see Methods), depending on the distance from the soma (Fig. 2e). We recorded the voltage at every branch of the model in response to a 5 ms step current input (Fig. 2f). Experimentally, such data could be obtained, for example, through voltage imaging [63].

We employed gradient descent to identify parameters that match this recording, with a regularizer that penalizes the difference between parameter values in neighboring branches, thereby targeting conductance profiles which vary smoothly across the dendritic tree [62, 64]. Despite the large number of parameters, gradient descent found a parameter set whose voltage response closely matched the observed voltage throughout the dendritic tree (Fig. 2g). To understand how much the whole cell voltage recording constrains the parameters [41], we used Bayesian inference (implemented with gradient-based Hamiltonian Monte–Carlo, details in Methods) to infer an ensemble of parameter sets all of which match the observed voltage. The resulting ensemble revealed regions along the dendritic tree at which the conductance profile was strongly constrained by the data (e.g., the transient sodium channel, Fig. 2h, left, e.g., around 400 µm). The parameter ensemble also revealed conductance profiles (e.g., axonal low-voltage activated calcium channel) which were only weakly constrained by the data, suggesting that diverse values of these parameters could lead to the desired activity (Fig. 2h, right, all posterior marginals in Supp. Fig. S5) [65]. Finally, we compared our method with an indicator-based genetic algorithm [18] and, as expected, we found that access to gradients leads to better convergence: While gradient descent converged to values of low loss within 100 iterations, the genetic algorithm had two orders of magnitude higher loss even after 500 iterations (Fig. 2i).

Nonlinear single neuron computation

In the two previous tasks, we showed that JAXLEY can learn model parameters such that simulations match recordings, and that it can do so as efficiently as state-of-the-art methods for small neural models and much more efficiently for large ones. Next, we demonstrate that gradient descent—commonly used to fit deep neural networks—can also be used to train single neurons to perform computational tasks.

We trained a single neuron model to solve a nonlinear pattern separation task. While point neurons cannot solve such tasks (because they linearly sum their inputs), it has been suggested that morphologically-detailed neurons might solve nonlinear pattern separation tasks in their dendritic tree [66–69]. While it has been demonstrated extensively that single neuron models respond nonlinearly to inputs [70–72], it has so far been difficult to train biophysically-detailed neurons on a particular task. Here, we show that stochastic gradient descent allows to train single neuron models with dendritic nonlinearities to perform nonlinear computations.

We defined a simple morphology consisting of a soma and two dendrites, and inserted sodium, potassium, and leak channels into all neurites of the cell. We then learned ion channel densities as well as length, radius, and axial resistivity of every compartment (72 parameters in total) for the neuron to have two outputs depending on the input: low somatic voltage (-70 mV), when both dendrites were stimulated with step currents of intermediate strength; high somatic voltage (35 mV), when one of the dendrites was stimulated strongly and the other one weakly (Fig. 2j). Therefore, the two classes were not linearly separable, requiring the neuron to perform a nonlinear computation.

After training the parameters with gradient descent, we found that the cell indeed learned to perform this task and spiked only when one dendrite was stimulated strongly (Fig. 2k), effectively having a nonlinear decision surface (Fig. 2l). We again compared our method to an indicator-based genetic algorithm and found that our method finds regions of lower loss more quickly than genetic algorithms (Fig. 2m).

Overall, these results show that gradient descent performs better than gradient-free methods in models with many parameters, opening up possibilities for studying at scale biophysical mechanisms throughout the full neuronal morphology.

Hybrid retina model of dendritic calcium measurements

So far, we have learned parameters of single neuron models using small datasets consisting of few stimulus/response pairs. Many models of neural systems, however, consist of multiple neurons potentially modelled at varying levels of detail, and datasets can contain thousands of stimulus/response pairs [16, 44, 74, 75]. Using a network model of the mouse retina, we demonstrate that JAXLEY allows to simultaneously infer cell-level and network-level parameters, and that it can identify parameters such that model simulations match datasets consisting of thousands of stimulus/response

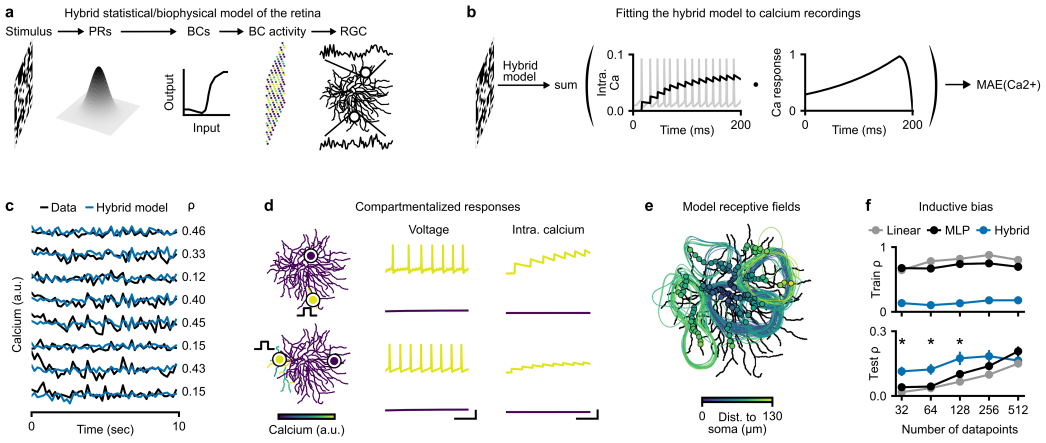


Figure 3. Hybrid model of the calcium responses of a retinal ganglion cell. (a) Schematic of experimental setup and hybrid model. A binary noise image (15 × 20 pixels) was presented to the retina of mice and dendritic calcium recordings were obtained with two-photon imaging [73]. We modelled photoreceptors (PR) as linear Gaussian filters, bipolar cells (BC) as point neurons with a non-linearity, and a retinal ganglion cell at full morphological detail with a variety of ion channels. (b) Schematic of training procedure and loss function. We stimulated the hybrid model with the noise image for 200 ms and recorded intracellular calcium concentration across all recording sites. We convolved this concentration with a calcium kernel and used the output at the last time step (200 ms) as input to a mean absolute error cost function to the experimental recording. (c) Measured and model-predicted calcium response across 50 noise images (200 ms each). (d) Left: Calcium response (colormap) of the trained hybrid model to a step-current to a single branch indicated by step current sketch. Middle: Voltage activity of the model at two branches, one at the stimulus site and one at a distant branch. Right: Intracellular calcium concentration in the same two recording sites. Scalebars: 50 ms, 30 mV, 0.025 mM. (e) Receptive fields of the hybrid model obtained in response to 1024 noise stimuli. (f) Pearson correlation coefficient between experimental data and model for train (top) and test (bottom) data, for a linear network, a multi-layer perceptron, and the hybrid model. Error bars show standard-error of mean over seven datasets (see Methods). Asterisk denotes statistically significant difference between mean correlations of hybrid model and MLP (one-sided t-test at p<0.05).

pairs.

As an example, we consider transient Off alpha retinal ganglion cells in the mouse retina, which show compartmentalized calcium signals in their dendrites in response to visual stimulation [73]. To understand the mechanistic underpinning of this behaviour, we built a hybrid model with statistical and mechanistic components: We modelled photoreceptors as convolution with a Gaussian filter, bipolar cells as point neurons with a nonlinearity [74], and a retinal ganglion cell (RGC) as a morphologically detailed biophysical neuron, with six different ion channels [76] distributed across its soma and dendritic tree (Fig. 3a). In order to model the recording of calcium signals, we convolved the intracellular calcium (from the calcium channel of the model) with a calcium kernel (Fig. 3b).

Using JAXLEY, we trained the hybrid model to predict dendritic calcium given checkerboard noise stimuli. The dataset consisted of 15,000 image-calcium pairs, with each image being presented for 200ms. We learned synaptic conductances from the bipolar cells onto the RGC (287 synaptic parameters), the radii of the branches of the retinal ganglion cell, the axial resistivities, as well as the somatic and basal membrane conductances (320 cell parameters). After training, we evaluated the trained model on a held-out test dataset. The model had a positive Pearson correlation coefficient with the experimental recording on 146 out of 147 recordings sites, with an average correlation of 0.25, and a maximum of 0.51 (Fig. 3c).

Next, we tested whether the hybrid model was able to predict experimentally measured phenomena which the model

had not been directly trained on. To test whether our model showed the same compartmentalized calcium responses as the measurements [73], we stimulated the trained model at a distal branch and recorded the model calcium response across all branches of the cell (Fig. 3d, left). We found that the calcium signal in response to local stimulation did not propagate through the entire cell, demonstrating a compartmentalized response of the model. While the neurons often exhibited high firing rates near the stimulation site in the dendrite (Fig. 3d, middle, [77, 78]), branches that were far away from the stimulation site showed no response (Fig. 3d, right). To further demonstrate the compartmentalized response of the hybrid model, we computed the receptive fields of all experimental recordings sites in response to noise images (Fig. 3e). The model receptive fields did not cover the entire cells and were roughly centered around the recording locations, qualitatively matching the receptive fields obtained from experimental measurements [73].

The mechanistic components of the model, including the anatomical structure, provide an inductive bias. Therefore, we investigated whether this inductive bias of the hybrid model could lead to better generalization to new data, especially when training data is scarce. We trained the hybrid model on reduced datasets of recordings from a single calcium scanfield and on a reduced set of images, and compared its performance to a linear model and a two-layer perceptron trained on the same datasets (Fig. 3f). For all models, we performed early stopping based on a validation set and evaluated the final performance on a held-out test set of 512 images. While the linear model and the perceptron performed better than the hybrid model on training data, the hybrid model performed better on held-out test data, when little training data was available. These results indicate that the inductive bias brought by the hybrid model effectively can limit the amount of overfitting in the model, and allow the model to have higher generalization performance than unconstrained artificial neural networks, at least in small data-regimes. These results suggest that hybrid components could be used as regularizers for deep neural networks models of neural systems [79–83].

Our results demonstrate that gradient descent allows to fit networks of biophysical neurons to large calcium datasets and allows to simultaneously learn cell-level and network-level parameters. The resulting models reproduce several experimental measurements and exhibit an inductive bias that leads to improved generalization performance on small datasets.

Biophysical recurrent network models can be trained to solve working memory tasks

To understand how computations are implemented in neural circuits, computational neuroscientists aim to train models to perform tasks [16, 45–48]. In particular, recurrent neural networks (RNNs) have been used to form hypotheses about population dynamics underlying cognition [15, 19, 45–47, 84, 85]. Typically, such RNNs consist of point neurons with rate-based or simplified spiking dynamics [42, 86], which prevents studying the contribution of channel dynamics or cellular processes [87]. We here show how JAXLEY makes is it possible to train biophysical models of neuronal networks to perform such tasks.

We implemented in JAXLEY an RNN consisting of Hodgkin–Huxley-type neurons with a simplified apical and basal dendrite, with each neuron equipped with a variety of voltage-gated ion channels [11]. We sparsely connected the recurrent network with conductance-based synapses [88] and obtained the outputs from passive readout units (Fig. 4a).

We first investigated the dynamics in this biophysical RNN before training. As with rate-based RNNs, these dynamics were strongly dependent on a global scaling factor (called 'gain') of all recurrent synaptic maximal conductances [89]. Our RNN transitioned from a stable to a chaotic regime when the gain was increased, with an intermediate region, where networks displayed regular firing (Fig. 4b, left). The ability of JAXLEY to perform automatic differentiation allowed us to quantify the stability of networks by numerically computing Lyapunov exponents [90, 91] (Supp. Fig. S7). When we increased the gain of the synapses, the maximal Lyapunov exponent transitions from a value smaller than 0, corresponding to stable dynamics, to a value larger than 0, corresponding to a chaotic system, where nearby trajectories diverge (Fig. 4b, right). This indicates that, upon parameter initialization, biophysical RNNs have similar dynamical regimes as rate-based RNNs.

We then trained the biophysical RNN to perform two working memory tasks, starting with a perceptual decision-making task requiring evidence integration over time [45, 47, 92–94]. We built a network of 20 recurrent neurons

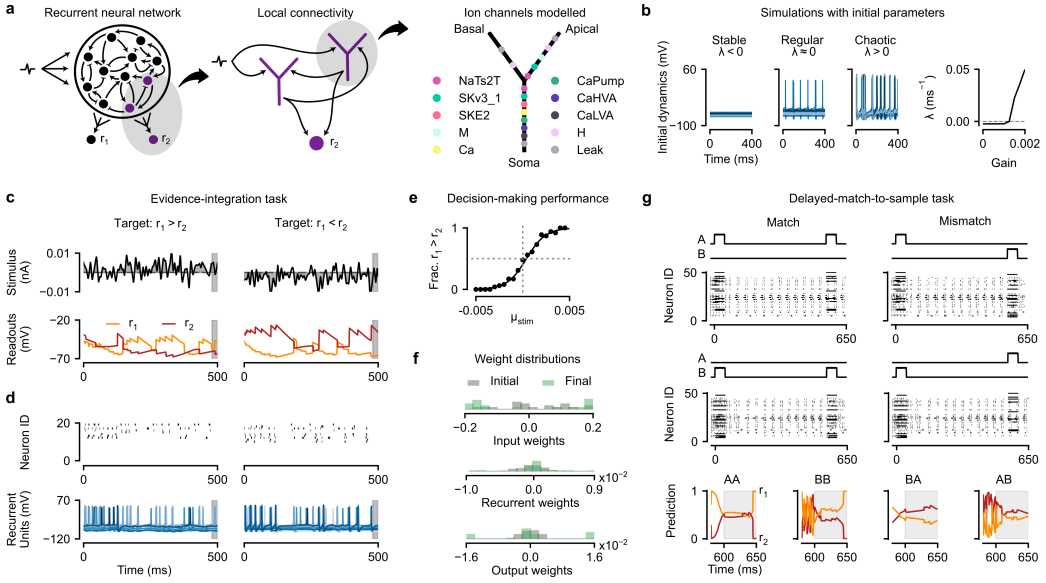


Figure 4. Recurrent neural network models with Hodgkin–Huxley-type neurons perform working memory tasks. (a) Left: Schematic of the RNN, with two recurrently connected neurons and one readout neuron highlighted. Neurons get stimulated at their basal dendrites and have recurrent synapses from somata to apical dendrites. Right: The ion channels modelled in each neuron. (b) Dynamics (left) and associated Lyapunov exponents (right) of the recurrently-connected neurons before learning the parameters and without any stimulus. The dynamics vary as a function of the scale (or 'gain') of the recurrent synaptic conductances. **Task 1** (c) Evidence integration task. We stimulated an RNN with a random Gaussian noise stimulus (top). After training, the RNN successfully learns to indicate whether the summed inputs are greater than zero (left) or less than zero (right), as indicated by voltage traces of the two readout neurons during the response period (grey). (d) Raster plots (upper) and voltage traces (lower) of the recurrently-connected neurons encoding the stimulus. (e) Psychometric curve showing the fraction of times the RNN reported the summed input to be greater than zero ($r_1 > r_2$), as a function of the stimulus mean (which modulates task difficulty). (f) Histogram of initial and trained input, recurrent, and output weights of the network. **Task 2** (g) Delayed-match-to-sample task. The RNN is presented with two stimuli (A/B) separated by a delay, and was trained to indicate whether the two stimuli 'match': The match case is where A or B contain both input step currents, and the mismatch case is where A and B each contain one input step current. Raster plots of the network activity, when presented with each of the four scenarios. Bottom row: Readout neuron prediction in the match (left, $r_1 > r_2$) and mismatch (right, $r_2 > r_1$) cases.

and stimulated each recurrent neuron with a noisy time series with either positive or negative mean value (Fig. 4c, top, see Methods). We trained input weights, recurrent weights, and readout weights (109 parameters) such that the network learned to differentiate between positive and negative stimuli during a response period after 500 ms. Despite the long time horizon of this task (500 ms, corresponding to 20k time steps of the simulation), gradient descent found parameters such that the RNN was able to perform the task (Fig. 4c, bottom), with the voltage in the readout neurons differentiating the input means with 99.9 % accuracy across 1,000 trials. The trained network showed sparse spiking activity with typically less than one third of neurons being active for a particular stimulus (Fig. 4d, top). Furthermore, the dynamics of the trained network without stimulus input were chaotic (Lyaponuv exponents close to 0, $\lambda = 2 \cdot 10^{-3}$, Fig. 4d, bottom), even though the initial dynamics were not ($\lambda = -1 \cdot 10^{-5}$), consistent with previous studies linking the regime close to 'the edge of chaos' to optimal computational performance [95, 96].

Next, we evaluated the generalization abilities of the trained biophysical RNN. We varied the mean value of the positive and negative stimuli and found that the relationship between average response and stimulus mean closely resembled the well-known sigmoidal psychometric curve of decision making, where the network more often failed when the stimulus had a lower signal-to-noise ratio (Fig. 4e). We also tested whether the network generalized in time and found that, despite having only been trained on tasks of 500 ms duration, the RNN could successfully solve evidence integration tasks of up to 3 seconds (Supp. Fig. S6). To solve these evidence integration tasks, some input, recurrent and output weights were pushed toward zero and the remaining weights were pushed toward their positive and negative constraints during training (Fig. 5f).

We next employed the RNN to solve a more challenging working memory task, a delayed match to sample task, where the RNN had to maintain information over an extended period of time [46, 47, 86, 97]. We trained the RNN to classify patterns, consisting of two step current inputs with a delay between them, into matching (same identity of the inputs) or non-matching (different identity of the inputs), i.e., a total of 4 input patterns (Fig. 4g, top). A central challenge in this task was that the network needed to memorise the identity of the first input within its dynamics for a delay period of 500 ± 50 ms until the second input was provided. We employed curriculum learning to solve this task: Starting from shorter delay periods of 100 ± 50 ms, we increased the delay period during training in steps of 100 ms [98]. By training input, recurrent, and readout weights, as well as synaptic time-constants of a network with 50 recurrent neurons (542 parameters), we found parameter sets which solved the task and correctly classified all four patterns (Fig. 4g, bottom).

Previous studies found that rate-based RNNs could learn working memory tasks either using transient coding or stable attractors [99–101]. We investigated which of those two mechanisms the biophysical RNN used to successfully maintain the stimulus identify during the delay period. To do so, we inspected the population dynamics of the trained biophysical RNN. The network had distinct responses to the identity of the first stimulus (Fig. 4g) and only when running the simulation much longer than the delay period the dynamics relaxed to the same attractor (Supp. Fig. S8). This suggests that the network used a form of transient coding to solve the task (depending on initialisation and training setup, other solutions might be possible [101]).

Overall, these results demonstrate that gradient descent allows training RNNs with biophysical detail to solve working memory tasks. This will allow a more quantitative investigation of the role of cellular mechanisms contributing to behavioral and cognitive computations.

Training biophysical networks with 100k parameters on large datasets

Finally, we show that gradient descent allows to train large biophysical models with thousands of cellular level and network-level parameters on machine learning-scale datasets to solve classical computer vision tasks like image recognition.

As a demonstration, we implemented a feedforward biophysical network model in JAXLEY and trained it to solve the classical MNIST task, without artificial nonlinearities such as ReLU activations. The network had three layers: The input and output layers consisted of neurons with ball-and-sticks morphologies and the hidden layer consisted of 64 morphologically detailed models obtained from reconstructions of CA1 cells (Fig. 5a) [43, 54]. The input and hidden layer had active ion channels (sodium, potassium, leak) in all branches, thereby permitting nonlinear computation as spike/no-spike decisions and allowing the network to have nonlinear responses. The output neurons only contained a leak channel and integrated hidden-layer activity. The network was interconnected by biophysical synapses [88] that could be excitatory or inhibitory. We trained sodium, potassium, and leak conductances of every branch in the circuit (55k parameters), as well as all synaptic weights (51k parameters).

We simulated the network for 10 ms, as this was the time it took for the stimulus to propagate through the network. After training with stochastic gradient descent (see Methods), upon being stimulated with a ‘0’ digit, the network propagated spikes through the first two layers, such that the somatic voltage of the output neuron corresponding to ‘0’ had high voltage after 10ms (Fig. 5b). When passed through a softmax, the output neuron activations indeed indicated a high probability for the digit ‘0’ (Fig. 5c). The network achieved an accuracy of 94.2 % on a held-out test dataset,

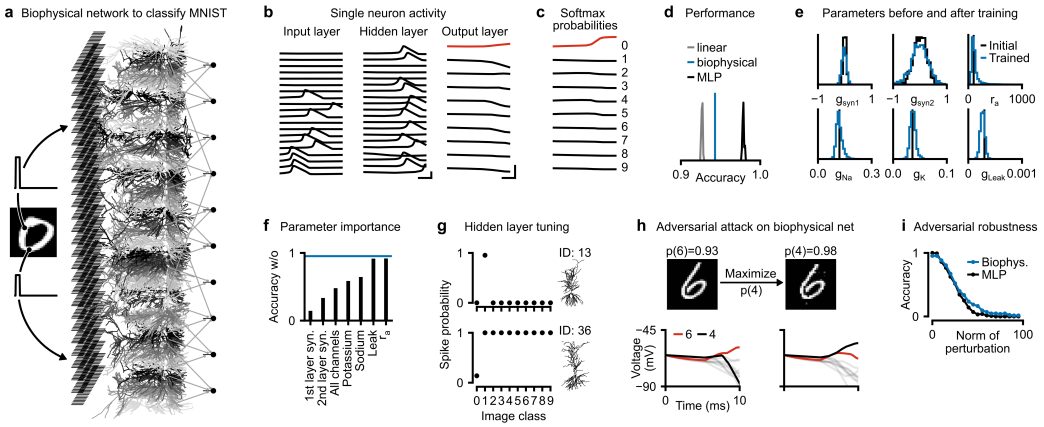


Figure 5. Training biophysically detailed networks to solve computer vision tasks. (a) We trained a biophysical network consisting of 28 × 28 input neurons, 64 morphologically detailed hidden neurons, and ten output neurons. We stimulated input neurons with step currents corresponding to MNIST pixel values and used voltages of the somata of output neurons after 10 ms as prediction of class membership. (b) Voltages measured at the somata of neurons in the trained network in response to an image labelled as '0'. Red color in the output layer indicates the image label. Scalebars: 2 ms and 80 mV. (c) Softmax probabilities computed from output voltages. (d) Histograms of test set accuracy of 50 linear networks (gray), 50 multi-layer perceptrons with 64 hidden neurons and ReLU activations (black), and the biophysical network (blue). (e) Histogram of parameters before (black) and after (blue) training. Training the network only leads to subtle shifts in parameter distributions. (f) Test set accuracy for trained network when subsets of parameters are reset to their initial value. Blue line is the full trained network. (g) Fraction of images that trigger a spike in two example hidden neurons across images from different classes. Top neuron has acquired ON-tuning for the digit '1', bottom neuron has OFF-tuning for the digit '0'. (h) Adversarial attack on the biophysical network. The trained biophysical network correctly classifies the image as a six with high confidence. After modifying the image with an adversarial attack, the image remains similar but the biophysical network confidently predicts a four. Bottom shows corresponding voltage traces of the output neurons (output neurons corresponding to other digits in gray). (i) Accuracy across 128 test set examples, as a function of the norm of the adversarial image perturbation.

which is higher than a linear classifier, demonstrating that the biophysical network successfully uses its nonlinearities to improve classification performance. The biophysical network, however, performed slightly worse than a multi-layer perceptron with ReLU nonlinearities, suggesting that the spike/no-spike nonlineariteits are either more difficult to train than ReLU nonlinearities, or that the (binary) spike/no-spike representations lead to lower bandwidth than graded ReLU activations (Fig. 5d).

How do the learned parameters of the biophysical network contribute to its ability to classify MNIST digits? Surprisingly, we found that the ranges of the trained synaptic parameters were roughly similar to the ranges of the untrained network and that the membrane channel conductances were roughly centered around their initial value (Fig. 5e). This does, however, not mean that the learned values of these parameters do not contribute to the learned network dynamics: When resetting subsets of parameters to their initial value, classification dropped substantially, sometimes reducing classification performance to chance level accuracy of 10 % (Fig. 5f). Resetting some parameters (e.g., leak conductances or axial resistivities) had a weaker but still substantial effect on performance (reduced accuracy to 91 %). Overall, although the average values of parameters remained roughly unchanged from initialization, the respective tuning of virtually all parameters in the system, including ion channel conductances, contributed to the learned network dynamics and classification performance. This indicates that biophysical simulations built purely from the aggregate statistics of measured parameter values could not be sufficient for the models to develop the ability to

perform tasks [11].

We then investigated which features the morphologically detailed hidden neurons were tuned to. Across 2560 images, we evaluated the fraction of images from each digit to trigger a spike in each hidden neuron. We found several neurons that were strongly tuned to individual digits: For example, we found a neuron which spiked for almost any '1' in the dataset, but rarely spiked for images of other digits in the dataset, suggesting that this cell had ON-tuning for the '1' digit, just one set of synapses away from the input layer (Fig. 5g, top). We also found neurons which exhibited OFF-tuning for some digits: For example, a hidden neuron spiked for any image which did not show a '0', but spiked only for a fraction of '0' images (Fig. 5g, bottom, tuning of all hidden neurons in Supp. Fig. S9). Neither of these neurons were tuned to individual digits before training (Supp. Fig. S10). These findings show that the trained biophysical network had hidden neurons strongly tuned to interpretable features, which emerged from training the network to solve a task.

Finally, efficient access to gradients does not only allow training biophysical models to match physiological data or perform a task, but it also opens up possibilities to perform new kinds of analyses on them, like evaluating adversarial robustness [102]. To illustrate this, we stimulated the biophysical network with an image showing a '6', and then altered minimally the image with gradient descent such that the network would classify it as a '4' (Fig. 5h). While the biophysical network classified the initial image as a '6' with high confidence, the perturbed image was classified as a '4'—despite only weak and barely noticeable changes to the image. We compared the adversarial robustness of the biophysical network to a trained multi-layer perceptron (MLP) with ReLU activations and the same number of hidden neurons. We found that the two networks are similarly vulnerable to adversarial attacks, and any improvement in adversarial robustness was comparably small, in contrast to previous studies suggesting that biophysical networks could largely improve adversarial robustness [54].

Overall, these results demonstrate that the ability of JAXLEY to perform backprop in biophysical models can be used to train large-scale networks at full biophysical detail. In particular, backprop overcomes virtually any computational limit on the number of parameters of biophysical networks that can be included in the optimization, thereby opening up new possibilities for task-trained biophysical models.

Discussion

Backpropagation of error ('backprop') and computational frameworks which provide efficient, scalable, and easy-to-use implementations, have been key to to the deep learning revolution. They have made it possible to efficiently optimise even very big systems with gradient descent. This, in turn, has made it possible to fully optimise entire machine learning workflows, and minimised the need for hand-designing ('feature engineering') its components. Several scientific disciplines are now adopting such 'differentiable programming' approaches in which entire pipelines are implemented as differentiable simulators and can therefore be optimised or fit to data [24–28, 103, 104]. However, for biophysical neuron models, currently used tools [31, 32, 105, 106] do not allow automatic differentiation. We here presented Jaxley, a new computational framework for differentiable simulation of neuroscience models with biophysical detail. Unlike previous biophysical simulation toolboxes, JAXLEY can perform automatic differentiation through its differential equation solver, thereby enabling backprop to compute the gradient with respect to virtually any biophysical parameter. We demonstrated that gradient descent allows to fit biophysical models of neural dynamics to large datasets of experimental voltage and calcium recordings and that it enables training biophysical models to perform physiologically meaningful computations, with as many as 100k parameters.

We expect that JAXLEY will enable a range of new investigations in neuroscience: First, it will make it possible to efficiently optimize detailed single-cell models. This will allow insights into cellular properties across cell types and their relationship with, for example, transcriptomic measurements [107–109], as well as into the contribution of dendritic processing to neural computations [70, 72, 110–113].

Second, it will facilitate creating large-scale biophysical network models. Such network models [11, 22, 114] have so far primarily been built in a bottom-up fashion: Models of single biophysical neurons are individually fit to electrophysiological recordings using genetic algorithms, with minimal adjustments of the network as a whole, and

computational properties are thought to emerge through this process. This approach has been successful in creating models which can reproduce aggregate statistics of neurophysiological measurements such as local field potentials [11, 115–117]. However, it is unlikely that such networks will be able to explain behavioural or cognitive computations. A prerequisite for this would be that these networks can perform relevant computational tasks such as image recognition or working memory tasks. However, a central impediment has been the lack of computational frameworks that allow to adjust (potentially thousands of) parameters of these models accurately and efficiently [10, 13, 18]. Inspired by the optimisation methods for deep neural networks which can fit millions of parameters with backprop and are accelerated by GPU parallelization, JAXLEY opens up possibilities to train parameters of large-scale biophysical network models with gradient descent. We note that, in our trained models, the *aggregate* distribution of parameters was very similar between trained and untrained models, again highlighting the difficulty of constructing task-performing networks from bottom-up considerations alone.

Third, in addition to training such models, JAXLEY will also enable numerous other applications: We showed how backprop enabled gradient-based Bayesian inference (Fig. 2h) [118, 119], and that it allowed us to investigate the stability of dynamical systems by computing their Lyapunov exponents (Fig. 4b), and made it possible to study adversarial attacks on biophysical network models (Fig. 5h). One will also be able to use gradients to compute maximally excitable stimuli [120, 121], or to design optimally discriminative experiments [122].

A central challenge in training biophysical models to perform computational tasks will be to bridge the timescales between biophysical mechanisms (milliseconds) and behavior (seconds). We showed that it is possible to fit data of up to one second length (Fig. 2d), but optimizing models on tasks that require backpropagating gradients along even longer simulations is challenging. In addition, opening up the possibility to learn the parameters of biophysical models with virtually no limit on the number of parameters will make mechanistic models prone to overfitting. Highly flexible models will also be more likely to identify *some* parameter sets that fit the data well, even if the model is wrong. To mitigate these issues, one should carefully design the loss function, the methods for model comparison, and the evaluation of the predictive quality on unseen data. One approach can be, for example, to train ensembles of models from different initial conditions, and to subsequently investigate which model-properties are robustly preserved across the ensemble [16].

JAXLEY contributes to a growing body of work on simulators for neuroscience, while offering key advantages for biophysically-detailed simulation. Like previous simulation toolboxes such as NEURON [105], GENESIS [32], or NEST [31], JAXLEY implements an implicit Euler solver required to solve the stiff dynamics of morphologically detailed biophysical neuron models (note that such a solver is not implemented in standard toolboxes for differential equations [52, 123]). Unlike these widely used neuroscience toolboxes, JAXLEY can automatically differentiate through this solver, thereby allowing to perform gradient descent without having to build differentiable emulators of biophysical models [72, 124]. In addition, its just-in-time compilation [125] and GPU parallelization capabilities [36–38, 54, 126] allow for fast simulation and training and enable scaling to large networks and datasets. By building upon the framework JAX [49], JAXLEY will further benefit from advances in the deep learning community at scaling and training large simulations. For example, for some tasks and training paradigms, forward mode automatic differentiation [127, 128] or evolutionary algorithms [42] have been reported to perform similarly to (or sometimes even better than) backprop. JAXLEY directly supports GPU-accelerated implementations of either of these algorithms, opening up possibilities to develop new methods for training of biophysical neural systems. JAXLEY has a flexible and easy-to-use interface and offers a growing and easily-extensible set of channels and synapses.

New experimental tools allow to measure connectivity [129, 130], morphology [131, 132], genetic identity [109], and activity [133] of neural circuits at increasing levels of detail and scale. JAXLEY is a powerful new tool which allows to integrate measurements of connectivity and morphology into biophysical simulations, while allowing the resulting networks to be fitted to data or computational tasks—just like deep neural networks. This will enable investigations of the biophysical basis of neural computation at unprecedented scales.

Acknowledgments

We thank Nathanael Bosch, Philipp Hennig, Sarah Müller, Seth Axen, Thomas Euler, Matthias Bethge, Thomas Zenkel, Federico D'Agostino, Jan-Matthis Lueckmann, and all members of our research groups for discussions. This work was supported by the German Research Foundation (DFG) through Germany's Excellence Strategy (EXC 2064 – Project number 390727645) and the CRC 1233 "Robust Vision", the German Federal Ministry of Education and Research (Tübingen AI Center, FKZ: 01IS18039A), the 'Certification and Foundations of Safe Machine Learning Systems in Healthcare' project funded by the Carl Zeiss Foundation, and the European Union (ERC, "DeepCoMechTome", ref. 101089288, "NextMechMod", ref. 101039115). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. MD, KLK, JB, MP, MG, and JKL are members of the International Max Planck Research School for Intelligent Systems (IMPRS-IS).

Author contributions

Conceptualization, Methodology: MD, PB, PJG, JHM. Software and Investigation: MD, KLK, MP, JB, MG, ZH, JKL, CS. Analysis: MD, KLK, MP, PB, PJG, JHM. Writing: MD, KLK, MP, PB, PJG, JHM. Writing (Review & Editing): JB, ZH, MG, JKL, CS. Funding acquisition: PB, JHM. Supervision: PB, PJG, JHM.

Methods

Code availability

Our toolbox, JAXLEY, is openly available at https://github.com/jaxleyverse/jaxley. Tutorials and examples of how to use the toolbox are available at https://jaxleyverse.github.io/jaxley. A collection of channels and synapses for use with JAXLEY is available at https://github.com/jaxleyverse/jaxley-mech. All code to generate results and figures is available at https://github.com/mackelab/jaxley_experiments.

Solving differential equations of biophysically detailed single neuron models

Below, we describe the numerical solver for the differential equations that define dynamics in morphologically detailed neuron models. We used an implicit Euler solver for the voltage equations, and an exponential Euler solver for the gates. We simulated voltage at internal nodes and, like NEURON, also simulated terminal nodes at branch points. To solve the tridiagonal system of equations of every branch, JAXLEY allows to choose between the Thomas algorithm [134] and the algorithm presented by Stone [135]. We used Stone's algorithm for all simulations.

We performed a leapfrog update of voltage equations and gate equations. As is also done in NEURON, at every time step, the current through mechanisms (channels and synapses) was evaluated twice at voltage values that differ by 0.001 mV. This allowed to infer the voltage-dependent and voltage-independent contributions to the current dynamics (which is required by the implicit Euler solver of the voltage equations). In order to achieve a high degree of parallelization, we modelled every branch in the cell (or network) with four compartments (with the exception of the network shown in Fig. 1f, for which we used two compartments). We split branches that were longer than 300 µm into sub-branches until each sub-branch was shorter than 300 µm.

Robust training of biophysical models

We use the following tools to improve training accuracy and robustness. First, we used parameter transformations that ensure that optimization of bounded parameters can be performed in unconstrained space and that biophysical parameters are on the same scale. In particular, we used an inverse sigmoid transformation $T(\theta) = -\log(\frac{1}{(\theta-l)/(u-l)} - 1)$, where l is the lower bound and u is the upper bound. Second, we used a variant of Polyak stochastic gradient descent [56, 136], which computes the step as step $= \gamma \frac{\nabla \mathcal{L}}{||\nabla \mathcal{L}||^\beta}$. This optimizer overcomes large variations of the gradient between steps. We used values of $\beta \in [0.8, 0.99]$. In some applications, we further multiplied the gradient with the loss value $\mathcal{L}$ [56, 136], such that the optimizer automatically reduced the learning rate towards the end of training. Third, when necessary, we used different optimizers for different types of parameters. This affects the Polyak gradient descent optimizer because it normalizes by the gradient norm. Fourth, we performed multi-level checkpointing to reduce the memory requirements of backpropagation through time. Fifth, when we observed vanishing or exploding gradients, we performed truncated backpropagation through time. We used this only for the hybrid mechanistic/statistical model of the retina. When performed, we interrupted gradient computation every 50 ms. All of these tools are implemented in JAXLEY.

Training overview

In Table 1, we list the training procedure for all tasks, including the number of optimized parameters, the number of branches that the model has (note that each branch is modelled with 4 compartments), the compute device we trained on, the number of gradient steps we took to arrive at the model shown in the figures, and the compute time it took to perform one gradient step.

Table 1. Overview of tasks and their training procedure.

Task	params	branches	device	gradient steps	time per step
L5PC synthetic (Fig. 2b)	19	339	CPU	10	5 sec
Allen cell 1 (Fig. 2d, left)	19	103	CPU	10	12 sec
Allen cell 2 (Fig. 2d, right)	19	99	CPU	10	12 sec
Fit conductance profiles (Fig. 2f)	1390	339	CPU	500	0.04 sec
Nonlinear neuron (Fig. 2g)	72	3	CPU	6400	0.0017 sec
Hybrid retina model (Fig. 3)	607	147	GPU, A100	560	55 sec
RNN for evidence integration (Fig. 4)	109	62	CPU	2000	12 sec
Delayed-match-to-sample task (Fig. 4)	459	152	GPU	1000	36 sec
Biophysical network for MNIST (Fig. 5)	105k	2000	GPU, V100	26k	28 sec

Fitting layer 5 pyramidal cells to somatic voltage recordings

Active mechanisms

We used two types of sodium channels, five types of potassium, two types of calcium, and a hyperpolarization-activated channel and inserted them in soma, basal and apical dendrites, and axon as done by Van Geit et al. [39].

Parameter initialization, parameter sharing, and parameter constraints

To fit the synthetic data shown in Fig. 2b, we optimized 19 parameters. These were the same parameters as used by Van Geit et al. [39] but our model did not contain a persistent sodium channel in the axon (following Deistler et al. [137], which demonstrated that using a persistent sodium channel is inconsistent with experimental measurements).

We used the same parameter search bounds as Van Geit et al. [39] for the synthetic data, but we enforced that somatic potassium existed (lower bound 0.25 mS/cm^2). For the experimental recordings from the Allen Cell Types Database, we used slightly larger bounds for two of the parameters to increase the flexibility of the model: We used an upper bound of 2 mS/cm^2 (instead of 1 mS/cm^2) and a lower bound of 10 ms for the delay of the calcium buffer (instead of 20 ms). We used a log-uniform distribution to initialize the delay of the calcium buffer. To fit recordings from the Allen Cell Types Database, we modified the leak conductance to $5{\cdot}10^{-5}$, 10^{-4}, 10^{-4}, and 10^{-4} mS/cm^2, respectively, leak reversal potential to -88, -88, -95, and -88 mV, capacitance to 2, 4, 3, and 2 $\mu F/cm^2$, respectively, and potassium reversal potential to -70 mV.

Summary statistics

Optimizing parameters of biophysical models with gradient descent requires that the loss function, and therefore also the summary statistics, are differentiable. This is not the case for typically used features such as spike count, which led us to defining different summary statistics.

For the synthetic data (Fig. 2b), we split the voltage trace into two windows and used the mean and standard deviation of these two windows as summary statistics. This led to a total of four summary statistics. To standardize the data, we divided the mean voltages by 8.0 and standard deviations by 4.0.

To fit data from the cell-types database, we used a set of four such windows. We defined the first window as the 20 ms after stimulus onset and used the maximal voltage of this window as summary statistic. This avoids that the model spikes before the experimental data. We defined the second window as a 15 ms window around the first spike in the experimental voltage and used the maximum value within this window as summary statistic. The third and fourth window spanned the next 60 and 905 ms and we used the mean and standard deviation during these windows. To standardize the summary statistics, we divided maximal voltages by 50.0, mean voltages by 10.0, and standard deviations by 5.0.

Training procedure

We used our variant of Polyak gradient descent with $\gamma = 1/3$ and $\beta = 0.8$ for the synthetic data and the experimental data. For both tasks, we use a mean absolute error loss to standardized summary statistics. For models based on the Allen Cell Types Database, we performed five initial simulations and initialized gradient descent runs at the parameter set which had lowest loss (among those five runs).

For both experiments, we trained for ten optimization steps. We also ran the genetic algorithm (IBEA-DEAP as implemented in the BluePyOpt package, version 1.14.11 [39]) for ten iterations, with ten simulations per iteration. We ran all experiments on an Apple MacBook Pro M3 CPU. For the synthetic data, a single gradient step took around five seconds, for the experimental data (which had a longer simulation time, but the neurons had fewer branches), it took around twelve seconds.

To compute how many simulations were needed for the genetic algorithm and gradient descent to obtain good solutions (Fig. 2c), we thresholded the loss value (see figure captions) and computed the number of simulations (or total compute time) divided by the number of converged runs.

Fitting voltage recordings of all branches

Active mechanisms

We used the same ion channels as for the task of fitting electrophysiology traces. To generate the observation, we sampled from a Gaussian process the conductance profiles as a function of the euclidean distance from the soma: we did this for the maximal conductances of all active ion channels in the apical dendrite (three active mechanisms) and the axon (seven active mechanisms).

Parameter initialization, parameter sharing, and parameter constraints

For the active mechanisms in the apical dendrite and the axon, we defined one parameter per branch in the cell and initialized parameters randomly and independently from each other. We used a single parameter for each mechanism in the soma and basal dendrite. Parameters had the same lower and upper bound as for the L5PC.

Training procedure

We used a mean absolute error loss function between the observed voltages and the simulated voltages, evaluated at every fifth time step between 1 ms and 5 ms of simulation time. We used our custom optimizer with $\beta = 0.8$, momentum 0.9, and learning rate 0.1. We trained the system for 500 iterations. We regularized the optimization such that neighboring branches had similar conductance values. In particular, we added to the loss the term $\lambda \cdot \sum_{b=1}^{\text{branches}} (\overline{g}_b - \text{parent}(\overline{g}_b))^2$, with regularization strength $\lambda = 0.001$.

One iteration of gradient descent took 0.04 seconds on an Apple MacBook Pro CPU.

Bayesian inference

To perform Bayesian inference over the parameters, we used a uniform prior over the free parameters with equivalent bounds as with the original training procedure. We also incorporated the same regularizer into the prior distribution. Since the optimization is performed in unconstrained space, we also conducted MCMC in this unconstrained space through a change of variables. We utilized a Laplace log-likelihood with a scale parameter of $\lambda = 0.001$, ensuring that the unnormalized posterior log density matched the loss function of the optimization problem.

We sampled from the posterior distribution with Hamiltonian Monte–Carlo (HMC) [138], which leverages the available gradient to efficiently sample from high-dimensional distributions. We used the BlackJAX [139] implementation of HMC. For each step, we performed five integration steps for the Hamiltonian dynamics with a step size of 0.01, leading to an average acceptance rate of $\approx 65\%$ (which indicates good performance for HMC [140]).

We initialized 200 chains with samples from the prior distribution. Each chain was run for 500 iterations in parallel on a single NVIDIA GeForce A100 GPU, and only the last sample of each chain was considered.

To visualize the learned conductance profile, we discretized distance from soma into eleven bins and grouped all parameters within each bin. We then calculated histograms for each of the bins and generated a spline interpolation of all quantile lines (Fig. 2h).

Nonlinear single neuron computation

Active mechanisms

The model contained sodium, potassium, and leak channels in all branches, with dynamics following the default implementation of the NEURON package [105].

Parameter initialization, parameter sharing, and parameter constraints

We trained sodium, potassium, and leak maximal conductances, as well as radius, length, and axial resistivity of every compartment in the model. Parameters were initialized randomly within uniform bounds. The bounds were [0.05, 1.1] for sodium, [0.01, 0.3] for potassium, and [0.0001, 0.001] for leak, all in mS/cm^2. The bounds for the radius were [0.1, 5.0] µm, for the length [1, 20] µm per compartment, and for the axial resistivity [500, 5500] Ωcm.

Training procedure

We used a mean absolute error loss function between the simulated voltages after 3 ms of simulated time and the class label (35 mV or -70 mV). We used the Adam optimizer with learning rate 0.01, and batch size 1. We trained the system for 200 epochs. One iteration of gradient descent took 1.7 ms on an Apple MacBook Pro CPU.

Hybrid model of the retina

Details on the data

Below, we describe the features of the experimental data which are most relevant to our training procedure (for full experimental details see Ran et al. [73]).

Images of 15 × 20 pixels were presented to a mouse retina. Each pixel had a size of 30 × 30 µm. Each image was presented for 200 ms, and a total of 1,500 images were presented. The images were centered on recording fields of the calcium activity. In total, 15 recording fields were made for the off alpha cell used in our work. Within each recording field, Ran et al. [73] defined a variable number of regions of interest, within which the calcium activity was recorded. In total, the data contained 232 regions of interest.

Data preprocessing

To denoise the calcium data, we lowpass-filtered the raw calcium data with a butterworth filter and a cutoff frequency of 7 Hz. We z-scored the resulting signal, with a different mean and standard deviation for each region of interest.

Next, we generated a single label for each image. We did this by performing a linear regression from image onto delayed calcium signals, and then used the calcium at the delay which was most predictive (i.e., had highest Pearson correlation between linear regression prediction and data). This led us to a delay of 1.8 seconds. As label, we used the low-pass filtered calcium value after this delay (starting from image onset).

Hybrid model

We modelled photoreceptors as a spatial linear Gaussian filter. The filter had a standard deviation of 50 µm. We modelled Bipolar cells as point neurons with a nonlinearity. The point neurons were spaced on a hexagonal grid with distance between neurons being 40 µm. The nonlinearity was taken from values measured by Schwartz et al. [74]. We connected every bipolar cell onto every branch of the retinal ganglion cell which was within 20 µm of the bipolar cell. Within every branch of the RGC, the synapse was made to the compartment which had the minimal Euclidean distance to the BC.

We computed the calcium activity of the model as the intracellular calcium value of the compartment that was the closest to the experimental recording site. We convolved this value with a double-exponential kernel to model dynamics of the calcium indicator. We used a rise-time of 5 ms and decay-time of 100 ms.

Active mechanisms in the retinal ganglion cell

Our model of the retinal ganglion cell contained six ion channels which were developed based on measurements from the retinal ganglion cells in cats [76]. The channels were a sodium channel, a leak channel, a delayed-rectifier potassium channel, a transient potassium channel, a calcium-dependent potassium channel, and a calcium channel. The model had all of these channels in every compartment of the model.

Parameter initialization, parameter sharing, and parameter constraints

We initialized all membrane conductances at previously published values [76], with an exception of sodium conductances which we initialized at 0.15 in the soma and 0.05 in the dendrite. We sampled the initial synaptic strengths randomly within 0 and 0.1 nS, and then divided the synaptic conductance by the number of postsynaptic connections that a bipolar cells makes (such that, in expectation, every BC has the same impact on the RGC). We initialized the axial resistivity of every compartment at 5,000 Ωcm. Finally, we initialized the radius of every dendritic compartment at 0.2 µm. We kept the diameter of the soma constant at 10 µm.

We trained the following set of parameters: One value for each maximal conductance in the soma (six parameters), one value for each maximal conductance in the dendrites, shared across all dendrites (six parameters), one value for each branch radius (147 parameters), one value for the axial resistivity of each branch (147 parameters), and one value for each synaptic conductance from the bipolar cells onto the retinal ganglion cells (250 parameters). In total, the model had 556 parameters.

We used the following bounds for optimization: For somatic conductances we used [0.05, 0.5] for sodium, [0.01, 0.1] for potassium, [10^{-5}, 10^{-3}] for leak, [0.01, 0.1] for transient potassium, [$2{\cdot}10^{-5}$, $2{\cdot}10^{-4}$] for calcium dependent potassium, and [0.002, 0.003] for calcium. All membrane conductance units are mS/cm^2. For dendritic conductances, we used the same bounds apart from the lower bound of 0 for sodium. For the branch radii, we used bounds of [0.1, 1.0] µm [73]. For the axial resistivities, we used [100, 10,000] Ωcm. For the synaptic conductances, we used [0.0, 0.2] nS.

Training procedure

We trained the model with Polyak stochastic gradient descent with momentum. We considered every kind of parameter (somatic conductance, dendritic conductance, radii, axial resistivities, and synaptic conductances) as separate parameters (which influences the computation of the gradient norm in Polyak stochastic gradient descent). We first trained the model for ten epochs with a learning rate of 0.01 and a momentum of 0.5. We used $\beta = 0.99$ to compute the norm in Polyak stochastic gradient descent.

We used a batchsize of 256 and used two level checkpointing to reduce the memory of backpropagation. To avoid vanishing or exploding gradients, we truncated the gradient in time. Specifically, we reset the gradient to 0 after 50, 100, and 150 ms of simulations.

We trained the model with mean absolute error loss between the experimentally measured (lowpass filtered) calcium value and the model predicted calcium value after 200 ms. Recording sites for which no recording was available in the data were masked out in the loss computation.

Receptive field computation

We followed Ran et al. [73] to compute receptive fields. We used automatic smoothness detection (ASD) [141] with 20 iterations of evidence optimization. We standardized all receptive fields to range from 0 to 1 and, for contours, thresholded the receptive fields at a value of 0.6.

Inductive bias

To evaluate the inductive bias of the hybrid model, we trained several such models, each on a reduced dataset. We reduced the dataset by using only a subset of stimulus/recording pairs from one scan field. We repeated this procedure over seven scan fields and five sizes of stimulus/recording pairs, namely 32, 64, 128, 256, 512. We trained each model with stochastic gradient descent as described above. To ensure that the gradient is stochastic (which can improve learning [142]), we used batchsizes of 4, 4, 8, 16, and 32 for the five dataset sizes, respectively. We performed early

stopping based on a validation set that contained 25 % of the training dataset and evaluated performance on 512 test datapoints. To avoid exceedingly long training times, we trained for at most 100 steps.

For the artificial neural network, we used a multi-layer perceptron with three hidden layers of [100, 100, 50] units and with ReLU activation functions.

A recurrent neural network of biophysical neurons performing an evidence integration task

Active mechanisms

All recurrent units used the same ion channels as for the task involving the layer 5 pyramidal cell, analogously inserted in soma, basal and apical dendrite. The two readout neurons were passive. The synapses were conductance-based, as described by Abbott and Marder [88].

Parameter initialization and parameter constraints

The recurrent units were connected with a probability of 0.2 by synapses from the soma of the presynaptic neurons to the apical dendrite of the postsynaptic neurons. All recurrent units had synapses onto the two readout neurons. 50 % of recurrent units had inhibitory outgoing synapses, created by setting their synaptic reversal potential to -75 mV, and the other half were excitatory with a synaptic reversal potential of 0 mV. Initial values for the maximum synaptic conductances were drawn from a standard normal distribution scaled by an initial gain g such that the bulk of the eigenspectrum of the synaptic weights lay in a circle on the complex plane with radius g (after multiplying inhibitory synapse weights by -1) [143]. We presented inputs by stimulating neurons at their basal dendrite. We set $g = 5.0\pi \ / \ 5 \cdot 10^3$, which is close to the transition point between stable and chaotic dynamics. For recurrent connections, we set the rate constant for transmitter-receptor dissociation rate (the $k\_$ parameter [65]), that influences the synaptic time-constants, to 1/1.0 ms. For connections to the rate-based readout neurons, we used slower synapses, with $k\_$ set to 1/40 ms. All recurrent units received stimulus input scaled by random initial weights drawn from $\mathcal{N}(0, 0.1)$.

We trained the maximal synaptic conductances constrained to the range [0, $3 \cdot \max(g)$]. We also trained the weights of the stimulus input to each neuron in the network restricted to the range [-0.2, 0.2]. The model had 109 trainable parameters.

Stimulus generation

Stimuli were generated by sampling values from normal distributions and then low-pass-filtering them with a maximum frequency cutoff of 2500 Hz. For training, we sampled values from $\mathcal{N}(\mu_-, 0.05)$ and $\mathcal{N}(\mu_+, 0.05)$, where $\mu_- \sim \mathcal{N}(-0.005, 0.0002)$ and $\mu_+ \sim \mathcal{N}(0.005, 0.0002)$.

Training procedure

We used a batch size of four, three levels of checkpointing, the Adam optimizer with a learning rate of 0.01, 2000 gradient steps, and gradient normalization with $\beta = 0.8$. We sampled new training data at every gradient step. Sweeps were used to inform the choice of hyperparameters. We used a cross entropy loss function with logits calculated as the mean readout activities in the last 20 ms of stimulus presentation. The training time was 7 hours on an Intel(R) Xeon(R) Silver 4116 CPU @ 2.10 GHz.

A recurrent neural network of biophysical neurons performing a delayed-match-to-sample task

Active mechanisms

We used the same channel mechanisms, compartments and synapses as in the evidence integration task.

Parameter initialization and parameter constraints

We used a network of 50 units (in line with previous work [47]), with a recurrent connectivity probability of 0.05, and all units connecting to the readout neurons. 80 % and 20 % of the neurons had excitatory and inhibitory outgoing connections, respectively. We set $g = 5.0\pi \ / \ 5 \cdot 10^3$. We set the $k\_$ parameter of recurrently connected neurons to

1, and to 0.1 for the slower synapses onto the readout neurons. The connection probability from input to (the basal dendrite of) recurrent units was 0.1, with initial weights drawn from $\mathcal{U}_{[0,1]}$.

We trained the maximal synaptic conductances, constrained to the range $[0, \infty)$, using a SoftPlus function. We also trained the weights of the stimulus input to each neuron in the network, restricted to the range [0, 4], as well as $k\_$ restricted to [0.05, 2]. This resulted in 459 parameters.

Stimulus generation

Stimuli consisted of square pulses with additive Gaussian noise sampled from $\mathcal{N}(0, 0.001)$. The onset period was 20 ms, and the stimulus and response durations were 50 ms. The delay period changed throughout the training procedure. Initial delay durations were drawn uniformly from $\mathcal{U}_{[50,150]}$. The average delay duration was increased in steps of 100 ms, whenever the network got at least 95 % accuracy on a single batch, till $\mathcal{U}_{[450,550]}$ was reached.

Training procedure

We used a batch size of 64, two levels of checkpointing, the Adam optimizer with a learning rate of 0.001, which was exponentially decayed to 0.0001 over 1000 epochs. We used a per time step cross entropy loss function, computed during the response period of the task. The training time was around 10 hours on an NVIDIA GeForce RTX 3090.

Computing Lyapunov exponents

We can quantify the stability of recurrent networks by measuring the average rate of divergence or convergence of nearby trajectories, which is given by the maximal Lyapunov exponent. To obtain the maximal Lyapunov exponent, we first discretised our model to obtain $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t)$, where $\mathbf{x}_t$ is a vector of all dynamic variables (e.g., voltages, gating variables) at time t, and $\mathbf{f}$ is one step of the chosen solver. We can then define the maximal Lyaponuv exponent as: $\lambda_{\max}(\mathbf{x}_0) = \lim_{t\to\infty} \frac{1}{t} \lim_{\epsilon\to 0} \log \frac{\|\epsilon \mathbf{u}_t\|}{\|\epsilon \mathbf{u}_0\|}$ where $\mathbf{u}_0$ is a perturbation to the initial state of the system $\mathbf{x}_0$. We measured the evolution of infinitesimal perturbations to $\mathbf{x}_{1:T}$ by calculating the Jacobian at each point along a trajectory.

We used the following numerical algorithm to approximate $\lambda_{\max}(\mathbf{x}_0)$ [90, 91]: First, we generated an initial state $\mathbf{x}_0$ and initial unit norm vector $\mathbf{q}_0$. After discarding initial transients for 4 seconds of simulation, we let the system $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t)$ and $\mathbf{q}_{t+1} = D\mathbf{f}|_{\mathbf{x}_t}\mathbf{q}_t$ (where D denotes the Jacobian) evolve for $T = 2.4 \cdot 10^5$ timestep (a further 6 seconds). Note that $\mathbf{q}_{t+1}$ can be efficiently computed using Jacobian vector products in JAX [49]. At every timestep we computed $r_t = \|\mathbf{q}_t\|$ and renormalised: $\mathbf{q}_t \leftarrow \frac{\mathbf{q}_t}{r_t}$. The maximal Lyaponuv exponent was then given by $\frac{1}{T}\sum_1^T \log \|r_t\|$.

The RNNs in Fig. 4b had 50 units with a 80% excitatory / 20% inhibitory split, and a connection probability of 0.2.

A biophysical network that performs computer vision tasks

Active mechanisms and synapses

The first layer consisted of 28 × 28 neurons, each stimulated by a step current whose amplitude was proportional to a pixel value. Each neuron in the first layer had a ball-and-stick morphology: Each cell consisted of four compartments, where one compartment (the soma) had a radius of 10 µm and all other compartments had a radius of 1 µm and a length of 10 µm. The input and hidden layer contained sodium, potassium, and leak channels in all branches, with dynamics following the default implementation of the NEURON package [105]. The output layer consisted of ten neurons with ball-and-stick morphologies (like the input layer neurons) and with leak dynamics. We used conductance-based synapses as described by Abbott and Marder [88]. The layers of the network were densely connected. We set the synaptic rate constant for transmitter-receptor dissociation to $k\_ = 1/4$.

Parameter initialization, parameter sharing, and parameter constraints

We optimized sodium, potassium, and leak maximal conductances of every branch in the network (50k parameters) and all synaptic conductances (50k parameters). We used the following bounds for the parameters: [0.05, 0.5] for sodium, [0.01, 0.1] for potassium, [0.0001, 0.001] for leak, [-5 / 28^2 / 25, 5 / 28^2 / 25] for synapses from the input to the hidden layer and [-5 / 64 / 25 / 2, 5 / 64 / 25 / 2] for synapses from the hidden layer to the output layer. We initialized

sodium maximal conductances at 0.12 mS/cm^2, potassium at 0.036 mS/cm^2, and leak at 0.0003 mS/cm^2. We initialized synaptic conductances as samples from a Gaussian distribution with mean 0 and standard deviation 1 / 28^2 / 25 for the first layer and standard deviation 1 / 64 / 25 / 2 for the second layer.

Training procedure

We used a batch size of 16 and cross entropy loss based on the values $(v + 65)/3$, where v is the somatic voltage of the output neurons after 10 ms of simulation. We used a cosine learning rate schedule and trained the network for seven epochs. Each gradient step took 25 seconds on a V100 GPU.

Adversarial attacks

To perform the adversarial attacks, we performed optimization of the input with gradient descent. We normalized every gradient step and used a learning rate of 5.0. We used bounds of [0, 1] for all pixel values during optimization. We used a cross entropy loss function. We chose the target label for the adversarial attack randomly, but ensured that the target label is not the true label. We computed accuracy based on 128 adversarial attacks.

References

[1] Andreas VM Herz, Tim Gollisch, Christian K Machens, and Dieter Jaeger. Modeling single-neuron dynamics and computations: a balance of detail and abstraction. *science*, 314(5796):80–85, 2006.

[2] Terrence J Sejnowski, Christof Koch, and Patricia S Churchland. Computational neuroscience. *Science*, 241(4871):1299–1306, 1988.

[3] Daniel Levenstein, Veronica A Alvarez, Asohan Amarasingham, Habiba Azab, Zhe S Chen, Richard C Gerkin, Andrea Hasenstaub, Ramakrishnan Iyer, Renaud B Jolivet, Sarah Marzen, Joseph D Monaco, Astrid A Prinz, Salma Quraishi, Fidel Santamaria, Sabyasachi Shivkumar, Matthew F Singh, Roger Traub, Farzan Nadim, Horacio G Rotstein, and A David Redish. On the role of theory and modeling in neuroscience. *The Journal of neuroscience: the official journal of the Society for Neuroscience*, 43(7):1074–1088, 2023.

[4] Geir Halnes, Torbjørn V Ness, Solveig Næss, Espen Hagen, Klas H Pettersen, and Gaute T Einevoll. *Electric Brain Signals: Foundations and Applications of Biophysical Modeling*. Cambridge University Press, 2024.

[5] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[6] Wilfrid Rall. Electrophysiology of a dendritic neuron model. *Biophysical journal*, 2(2 Pt 2):145, 1962.

[7] Christof Koch, Tomaso Poggio, and Vincent Torre. Retinal ganglion cells: a functional interpretation of dendritic morphology. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 298(1090):227–263, 1982.

[8] Astrid A. Prinz, Cyrus P. Billimoria, and Eve Marder. Alternative to hand-tuning conductance-based models: Construction and analysis of databases of model neurons. *Journal of Neurophysiology*, 90(6):3998–4015, 2003.

[9] M Pospischil, M Toledo-Rodriguez, C Monier, Z Piwkowska, T Bal, Y Frégnac, H Markram, and A Destexhe. Minimal hodgkin-huxley type models for different classes of cortical and thalamic neurons. *Biological Cybernetics*, 99(4-5), 2008.

[10] Etay Hay, Sean Hill, Felix Schürmann, Henry Markram, and Idan Segev. Models of neocortical layer 5b pyramidal cells capturing a wide range of dendritic and perisomatic active properties. *PLoS computational biology*, 7(7):e1002107, 2011.

[11] Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, Selim Arsever, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.

[12] Kanaka Rajan, Christopher D Harvey, and David W Tank. Recurrent network models of sequence generation and memory. *Neuron*, 90(1):128–142, 2016.

[13] Nathan W Gouwens, Jim Berg, David Feng, Staci A Sorensen, Hongkui Zeng, Michael J Hawrylycz, Christof Koch, and Anton Arkhipov. Systematic generation of biophysically detailed models for diverse cortical neuron types. *Nature communications*, 9 (1):710, 2018.

[14] Valerio Mante, David Sussillo, Krishna V Shenoy, and William T Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *nature*, 503(7474):78–84, 2013.

[15] Saurabh Vyas, Matthew D. Golub, David Sussillo, and Krishna V. Shenoy. Computation through neural population dynamics. *Annual Review of Neuroscience*, 43(Volume 43, 2020):249–275, 2020.

[16] Janne K Lappalainen, Fabian D Tschopp, Sridhama Prakhya, Mason McGill, Aljoscha Nern, Kazunori Shinomiya, Shin-ya Takemura, Eyal Gruntman, Jakob H Macke, and Srinivas C Turaga. Connectome-constrained deep mechanistic networks predict neural responses across the fly visual system at single-neuron resolution. *bioRxiv*, pages 2023–03, 2023.

[17] Wulfram Gerstner, Henning Sprekeler, and Gustavo Deco. Theory and simulation in neuroscience. *science*, 338(6103):60–65, 2012.

[18] Werner Van Geit, Erik De Schutter, and Pablo Achard. Automated neuron model optimization techniques: a review. *Biological cybernetics*, 99:241–251, 2008.

[19] David Sussillo, Mark M Churchland, Matthew T Kaufman, and Krishna V Shenoy. A neural network that finds a naturalistic solution for the production of muscle activity. *Nature Neuroscience*, 18(7):1025–1033, 2015.

[20] Willem AM Wybo, Jakob Jordan, Benjamin Ellenberger, Ulisses Marti Mengual, Thomas Nevian, and Walter Senn. Data-driven reduction of dendritic morphologies with preserved dendro-somatic responses. *Elife*, 10:e60936, 2021.

[21] Michalis Pagkalos, Spyridon Chavlis, and Panayiota Poirazi. Introducing the dendrify framework for incorporating dendrites to spiking neural networks. *Nature Communications*, 14(1):131, 2023.

[22] Michael W Reimann, Sirio Bolaños-Puchet, Jean-Denis Courcol, Daniela Egas Santander, Alexis Arnaudon, Benoît Coste, Thomas Delemontex, Adrien Devresse, Hugo Dictus, Alexander Dietz, et al. Modeling and simulation of rat non-barrel somatosensory cortex. part i: Modeling anatomy. *bioRxiv*, pages 2022–08, 2022.

[23] Gaute T Einevoll, Alain Destexhe, Markus Diesmann, Sonja Grün, Viktor Jirsa, Marc de Kamps, Michele Migliore, Torbjørn V Ness, Hans E Plesser, and Felix Schürmann. The scientific case for brain simulations. *Neuron*, 102(4):735–744, 2019.

[24] Samuel Schoenholz and Ekin Dogus Cubuk. Jax md: a framework for differentiable physics. *Advances in Neural Information Processing Systems*, 33:11428–11441, 2020.

[25] Philipp Holl, Nils Thuerey, and Vladlen Koltun. Learning to control pdes with differentiable physics. In *International Conference on Learning Representations*, 2020.

[26] Mohammed AlQuraishi and Peter K Sorger. Differentiable biology: using deep learning for biophysics-based and data-driven modeling of molecular mechanisms. *Nature methods*, 18(10):1169–1180, 2021.

[27] Tommaso Dorigo, Andrea Giammanco, Pietro Vischia, Max Aehle, Mateusz Bawaj, Alexey Boldyrev, Pablo de Castro Manzano, Denis Derkach, Julien Donini, Auralee Edelen, et al. Toward the end-to-end optimization of particle physics instruments with differentiable programming. *Reviews in Physics*, page 100085, 2023.

[28] Chaopeng Shen, Alison P Appling, Pierre Gentine, Toshiyuki Bandai, Hoshin Gupta, Alexandre Tartakovsky, Marco Baity-Jesi, Fabrizio Fenicia, Daniel Kifer, Li Li, et al. Differentiable modelling to unify machine learning and physical models for geosciences. *Nature Reviews Earth & Environment*, 4(8):552–567, 2023.

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[31] Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

[32] James M Bower and David Beeman. *The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System*. Springer Science & Business Media, 2012.

[33] Michael L Hines and Nicholas T Carnevale. The neuron simulation environment. *Neural computation*, 9(6):1179–1209, 1997.

[34] Nicholas T Carnevale and Michael L Hines. *The NEURON book*. Cambridge University Press, 2006.

[35] Michael Hines, Andrew P Davison, and Eilif Muller. Neuron and python. *Frontiers in neuroinformatics*, 3:391, 2009.

[36] Pramod Kumbhar, Michael Hines, Jeremy Fouriaux, Aleksandr Ovcharenko, James King, Fabien Delalondre, and Felix Schürmann. Coreneuron: an optimized compute engine for the neuron simulator. *Frontiers in neuroinformatics*, 13:63, 2019.

[37] Denis Alevi, Marcel Stimberg, Henning Sprekeler, Klaus Obermayer, and Moritz Augustin. Brian2cuda: flexible and efficient simulation of spiking neural network models on gpus. *Frontiers in Neuroinformatics*, 16:883700, 2022.

[38] Roy Ben-Shalom, Alexander Ladd, Nikhil S Artherya, Christopher Cross, Kyung Geun Kim, Hersh Sanghevi, Alon Korngreen, Kristofer E Bouchard, and Kevin J Bender. Neurogpu: Accelerating multi-compartment, biophysically detailed neuron simulations on gpus. *Journal of neuroscience methods*, 366:109400, 2022.

[39] Werner Van Geit, Michael Gevaert, Giuseppe Chindemi, Christian Rössert, Jean-Denis Courcol, Eilif B Muller, Felix Schürmann, Idan Segev, and Henry Markram. Bluepyopt: leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. *Frontiers in neuroinformatics*, 10:17, 2016.

[40] Pedro J Gonçalves, Jan-Matthis Lueckmann, Michael Deistler, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F Podlaski, Sara A Haddad, Tim P Vogels, et al. Training deep neural density estimators to identify mechanistic models of neural dynamics. *Elife*, 9:e56261, 2020.

[41] Ilenna Simone Jones and Konrad Paul Kording. Efficient optimization of ode neuron models using gradient descent. *arXiv preprint arXiv:2407.04025*, 2024.

[42] James Hazelden, Yuhan Helena Liu, Eli Shlizerman, and Eric Shea-Brown. Evolutionary algorithms as an alternative to backpropagation for supervised training of biophysical neural networks and neural odes. *arXiv preprint arXiv:2311.10869*, 2023.

[43] Gowri K Pyapali, Attila Sik, Markku Penttonen, Gyorgy Buzsaki, and Dennis A Turner. Dendritic properties of hippocampal ca1 pyramidal neurons in the rat: intracellular staining in vivo and in vitro. *Journal of Comparative Neurology*, 391(3):335–352, 1998.

[44] Cornelius Schröder, David Klindt, Sarah Strauss, Katrin Franke, Matthias Bethge, Thomas Euler, and Philipp Berens. System identification with biophysical constraints: A circuit model of the inner retina. *Advances in Neural Information Processing Systems*, 33:15439–15450, 2020.

[45] H. Francis Song, Guangyu R. Yang, and Xiao-Jing Wang. Training excitatory-inhibitory recurrent neural networks for cognitive tasks: A simple and flexible framework. *PLOS Computational Biology*, 12(2):1–30, 02 2016.

[46] Guangyu Robert Yang, Madhura R. Joglekar, H. Francis Song, William T. Newsome, and Xiao-Jing Wang. Task representations in neural networks trained to perform many cognitive tasks. *Nature Neuroscience*, 22:297 – 306, 2019.

[47] Alexis Dubreuil, Adrian Valente, Manuel Beiran, Francesca Mastrogiuseppe, and Srdjan Ostojic. The role of population structure in computations through neural dynamics. *Nature Neuroscience*, 25(6):783–794, Jun 2022.

[48] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

[49] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[50] Michael Hines. Efficient computation of branched nerve equations. *International journal of bio-medical computing*, 15(1): 69–76, 1984.

[51] Patrick Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.

[52] Chaoming Wang, Tianqiu Zhang, Xiaoyu Chen, Sichao He, Shangyang Li, and Si Wu. Brainpy, a flexible, integrative, efficient, and extensible framework for general-purpose brain dynamics programming. *Elife*, 12, 2023.

[53] Jonas Beck, Nathanael Bosch, Michael Deistler, Kyra L. Kadhim, Jakob H. Macke, Philipp Hennig, and Philipp Berens. Diffusion tempering improves parameter estimation with probabilistic integrators for ordinary differential equations. In *Forty-first International Conference on Machine Learning*, 2024.

[54] Yichen Zhang, Gan He, Lei Ma, Xiaofei Liu, JJ Johannes Hjorth, Alexander Kozlov, Yutao He, Shenjian Zhang, Jeanette Hellgren Kotaleski, Yonghong Tian, et al. A gpu-based computational framework that bridges neuron simulation and artificial intelligence. *Nature Communications*, 14(1):5798, 2023.

[55] Andreas Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and software*, 1(1):35–54, 1992.

[56] Nicolas Loizou, Sharan Vaswani, Issam Hadj Laradji, and Simon Lacoste-Julien. Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence. In *International Conference on Artificial Intelligence and Statistics*, pages 1306–1314. PMLR, 2021.

[57] Giorgio A Ascoli, Duncan E Donohue, and Maryam Halavi. Neuromorpho. org: a central resource for neuronal morphologies. *Journal of Neuroscience*, 27(35):9247–9251, 2007.

[58] Allen Institute for Brain Science. Allen cell types database. http://celltypes.brain-map.org/, 2016.

[59] Yingbo Ma, Vaibhav Dixit, Michael J Innes, Xingjian Guo, and Chris Rackauckas. A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9. IEEE, 2021.

[60] Quentin JM Huys, Misha B Ahrens, and Liam Paninski. Efficient estimation of detailed single-neuron models. *Journal of neurophysiology*, 96(2):872–890, 2006.

[61] Jeffrey C Magee and Daniel Johnston. Characterization of single voltage-gated na+ and ca2+ channels in apical dendrites of rat ca1 pyramidal neurons. *The Journal of physiology*, 487(1):67–90, 1995.

[62] Michele Migliore and Gordon M Shepherd. Emerging rules for the distributions of active dendritic conductances. *Nature Reviews Neuroscience*, 3(5):362–370, 2002.

[63] Yongxian Xu, Peng Zou, and Adam E Cohen. Voltage imaging with genetically encoded indicators. *Current Opinion in Chemical Biology*, 39:1–10, 2017.

[64] Jeffrey C Magee. Dendritic hyperpolarization-activated currents modify the integrative properties of hippocampal ca1 pyramidal neurons. *Journal of Neuroscience*, 18(19):7613–7624, 1998.

[65] Astrid A Prinz, Dirk Bucher, and Eve Marder. Similar network activity from disparate circuit parameters. *Nature neuroscience*, 7(12):1345–1352, 2004.

[66] Stephen R Williams and Greg J Stuart. Dependence of epsp efficacy on synapse location in neocortical pyramidal neurons. *Science*, 295(5561):1907–1910, 2002.

[67] Alon Polsky, Bartlett W Mel, and Jackie Schiller. Computational subunits in thin dendrites of pyramidal cells. *Nature neuroscience*, 7(6):621–627, 2004.

[68] Attila Losonczy and Jeffrey C Magee. Integrative properties of radial oblique dendrites in hippocampal ca1 pyramidal neurons. *Neuron*, 50(2):291–307, 2006.

[69] Greg J Stuart and Nelson Spruston. Dendritic integration: 60 years of progress. *Nature neuroscience*, 18(12):1713–1721, 2015.

[70] Panayiota Poirazi, Terrence Brannon, and Bartlett W Mel. Pyramidal neuron as two-layer neural network. *Neuron*, 37(6): 989–999, 2003.

[71] Michael London and Michael Häusser. Dendritic computation. *Annu. Rev. Neurosci.*, 28(1):503–532, 2005.

[72] David Beniaguev, Idan Segev, and Michael London. Single cortical neurons as deep artificial neural networks. *Neuron*, 109 (17):2727–2739, 2021.

[73] Yanli Ran, Ziwei Huang, Tom Baden, Timm Schubert, Harald Baayen, Philipp Berens, Katrin Franke, and Thomas Euler. Type-specific dendritic integration in mouse retinal ganglion cells. *Nature Communications*, 11(1):2101, 2020.

[74] Gregory W Schwartz, Haruhisa Okawa, Felice A Dunn, Josh L Morgan, Daniel Kerschensteiner, Rachel O Wong, and Fred Rieke. The spatial structure of a nonlinear receptive field. *Nature neuroscience*, 15(11):1572–1580, 2012.

[75] Xu Han, Ben Vermaercke, and Vincent Bonin. Diversity of spatiotemporal coding reveals specialized visual processing streams in the mouse cortex. *Nature communications*, 13(1):3249, 2022.

[76] JF Fohlmeister and RF Miller. Impulse encoding mechanisms of ganglion cells in the tiger salamander retina. *Journal of neurophysiology*, 78(4):1935–1947, 1997.

[77] Toby J Velte and Richard H Masland. Action potentials in the dendrites of retinal ganglion cells. *Journal of neurophysiology*, 81(3):1412–1417, 1999.

[78] Benjamin Sivyer and Stephen R Williams. Direction selectivity is computed by active dendritic integration in retinal ganglion cells. *Nature neuroscience*, 16(12):1848–1856, 2013.

[79] Lane McIntosh, Niru Maheswaranathan, Aran Nayebi, Surya Ganguli, and Stephen Baccus. Deep learning models of the retinal response to natural scenes. *Advances in neural information processing systems*, 29, 2016.

[80] Saad Idrees, Michael B Manookin, Fred Rieke, Greg D Field, and Joel Zylberberg. Biophysical neural adaptation mechanisms enable artificial neural networks to capture dynamic retinal computation. *Nature Communications*, 15(1):5957, 2024.

[81] Yongrong Qiu, David A Klindt, Klaudia P Szatko, Dominic Gonschorek, Larissa Hoefling, Timm Schubert, Laura Busse, Matthias Bethge, and Thomas Euler. Efficient coding of natural scenes improves neural system identification. *PLoS computational biology*, 19(4):e1011037, 2023.

[82] Danny da Costa, Lukas Kornemann, Rainer Goebel, and Mario Senden. Convolutional neural networks develop major organizational principles of early visual cortex when enhanced with retinal sampling. *Scientific Reports*, 14(1):8980, 2024.

[83] Jack Lindsey, Samuel A. Ocko, Surya Ganguli, and Stephane Deny. A unified theory of early visual representations from retina to cortex through anatomically constrained deep cnns. In *International Conference on Learning Representations*, 2019.

[84] Omri Barak. Recurrent neural networks as versatile tools of neuroscience research. *Current Opinion in Neurobiology*, 46:1–6, 2017. Computational Neuroscience.

[85] Matthijs Pals, Jakob H. Macke, and Omri Barak. Trained recurrent neural networks develop phase-locked limit cycles in a working memory task. *PLOS Computational Biology*, 20(2):1–23, 02 2024.

[86] Robert Kim and Terrence J. Sejnowski. Strong inhibitory signaling underlies stable temporal dynamics and working memory in spiking neural networks. *Nature Neuroscience*, 24(1):129–139, 2021.

[87] Lukas N Groschner, Laura Chan Wah Hak, Rafal Bogacz, Shamik DasGupta, and Gero Miesenböck. Dendritic integration of sensory evidence in perceptual decision-making. *Cell*, 173(4):894–905, 2018.

[88] LF Abbott and Eve Marder. Modeling small networks, 1998.

[89] H. Sompolinsky, A. Crisanti, and H. J. Sommers. Chaos in random neural networks. *Phys. Rev. Lett.*, 61:259–262, Jul 1988.

[90] Giancarlo Benettin, L. Galgani, Antonio Giorgilli, and Jean-Marie Strelcyn. Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems - a method for computing all of them. i - theory. ii - numerical application. *Meccanica*, 15:21–30, 03 1980.

[91] Rainer Engelken, Fred Wolf, and L. F. Abbott. Lyapunov spectra of chaotic recurrent neural networks. *Phys. Rev. Res.*, 5: 043044, Oct 2023.

[92] William T Newsome, Kenneth H Britten, and J Anthony Movshon. Neuronal correlates of a perceptual decision. *Nature*, 341 (6237):52–54, 1989.

[93] Bingni W Brunton, Matthew M Botvinick, and Carlos D Brody. Rats and humans can optimally accumulate evidence for decision-making. *Science*, 340(6128):95–98, 2013.

[94] Benjamin B Scott, Christine M Constantinople, Athena Akrami, Timothy D Hanks, and Carlos B Brody. Fronto-parietal cortical circuits encode accumulated evidence with a diversity of timescales. *Neuron*, 95(2):385–398, 2017.

[95] Nils Bertschinger and Thomas Natschläger. Real-Time Computation at the Edge of Chaos in Recurrent Neural Networks. *Neural Computation*, 16(7):1413–1436, 07 2004. ISSN 0899-7667.

[96] Robert Legenstein and Wolfgang Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323–334, 2007. ISSN 0893-6080.

[97] Joaquin M. Fuster and Garrett E. Alexander. Neuron activity related to short-term memory. *Science*, 173(3997):652–654, 1971.

[98] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[99] Omri Barak, David Sussillo, Ranulfo Romo, Misha Tsodyks, and L.F. Abbott. From fixed points to chaos: Three models of delayed discrimination. *Progress in Neurobiology*, 103:214–222, 2013. Conversion of Sensory Signals into Perceptions, Memories and Decisions.

[100] A. Emin Orhan and Wei Ji Ma. A diverse range of factors affect the nature of neural representations underlying short-term memory. *Nature Neuroscience*, 22(2):275–283, 2019.

[101] Elia Turner, Kabir V Dabholkar, and Omri Barak. Charting and navigating the space of solutions for recurrent neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 25320–25333. Curran Associates, Inc., 2021.

[102] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations*, 2017.

[103] Bharath Ramsundar, Dilip Krishnamurthy, and Venkatasubramanian Viswanathan. Differentiable physics: A position piece. *arXiv preprint arXiv:2109.07573*, 2021.

[104] Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023.

[105] Michael L Hines and Nicholas T Carnevale. Neuron: a tool for neuroscientists. *The neuroscientist*, 7(2):123–135, 2001.

[106] Dan FM Goodman and Romain Brette. The brian simulator. *Frontiers in neuroscience*, 3:643, 2009.

[107] Anirban Nandi, Thomas Chartrand, Werner Van Geit, Anatoly Buchin, Zizhen Yao, Soo Yeun Lee, Yina Wei, Brian Kalmbach, Brian Lee, Ed Lein, et al. Single-neuron models linking electrophysiology, morphology, and transcriptomics across cortical cell types. *Cell reports*, 40(6), 2022.

[108] Yves Bernaerts, Michael Deistler, Pedro J Gonçalves, Jonas Beck, Marcel Stimberg, Federico Scala, Andreas S Tolias, Jakob Macke, Dmitry Kobak, and Philipp Berens. Combined statistical-mechanistic modeling links ion channel genes to physiology of cortical neuron types. *bioRxiv*, pages 2023–03, 2023.

[109] Federico Scala, Dmitry Kobak, Matteo Bernabucci, Yves Bernaerts, Cathryn René Cadwell, Jesus Ramon Castro, Leonard Hartmanis, Xiaolong Jiang, Sophie Laturnus, Elanine Miranda, et al. Phenotypic variation of transcriptomic cell types in mouse motor cortex. *Nature*, 598(7879):144–150, 2021.

[110] Christof Koch. *Biophysics of computation: information processing in single neurons*. Oxford university press, 2004.

[111] Albert Gidon, Timothy Adam Zolnik, Pawel Fidzinski, Felix Bolduan, Athanasia Papoutsi, Panayiota Poirazi, Martin Holtkamp, Imre Vida, and Matthew Evan Larkum. Dendritic action potentials and computation in human layer 2/3 cortical neurons. *Science*, 367(6473):83–87, 2020.

[112] Panayiota Poirazi and Athanasia Papoutsi. Illuminating dendritic function with computational models. *Nature reviews neuroscience*, 21(6):303–321, 2020.

[113] Spyridon Chavlis and Panayiota Poirazi. Drawing inspiration from biological dendrites to empower artificial neural networks. *Current opinion in neurobiology*, 70:1–10, 2021.

[114] James B Isbister, András Ecker, Christoph Pokorny, Sirio Bolaños-Puchet, Daniela Egas Santander, Alexis Arnaudon, Omar Awile, Natali Barros-Zulaica, Jorge Blanco Alonso, Elvis Boci, et al. Modeling and simulation of neocortical micro-and mesocircuitry. part ii: Physiology and experimentation. *bioRxiv*, pages 2023–05, 2023.

[115] Henrik Lindén, Espen Hagen, Szymon Łęski, Eivind S Norheim, Klas H Pettersen, and Gaute T Einevoll. Lfpy: a tool for biophysical simulation of extracellular potentials generated by detailed model neurons. *Frontiers in neuroinformatics*, 7:41, 2014.

[116] Tobias C Potjans and Markus Diesmann. The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. *Cerebral cortex*, 24(3):785–806, 2014.

[117] Sacha J Van Albada, Andrew G Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan B Stokes, David R Lester, Markus Diesmann, and Steve B Furber. Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software nest for a full-scale cortical microcircuit model. *Frontiers in neuroscience*, 12:291, 2018.

[118] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*, 2017.

[119] Dieterich Lawson, Allan Raventós, Andrew Warrington, and Scott Linderman. Sixo: Smoothing inference with twisted objectives. *Advances in Neural Information Processing Systems*, 35:38844–38858, 2022.

[120] Edgar Y Walker, Fabian H Sinz, Erick Cobos, Taliah Muhammad, Emmanouil Froudarakis, Paul G Fahey, Alexander S Ecker, Jacob Reimer, Xaq Pitkow, and Andreas S Tolias. Inception loops discover what excites neurons most using deep predictive models. *Nature neuroscience*, 22(12):2060–2065, 2019.

[121] Pouya Bashivan, Kohitij Kar, and James J DiCarlo. Neural population control via deep image synthesis. *Science*, 364(6439), 2019.

[122] Max F Burg, Thomas Zenkel, Michaela Vystrčilová, Jonathan Oesterle, Larissa Höfling, Konstantin Friedrich Willeke, Jan Lause, Sarah Müller, Paul G Fahey, Zhiwei Ding, et al. Maximally discriminative stimuli for functional cell type identification. In *The Twelfth International Conference on Learning Representations*, 2023.

[123] Chris Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. Diffeqflux.jl - a julia library for neural differential equations. *arXiv preprint arXiv:1902.02376*, 2019.

[124] Jonas Verhellen, Kosio Beshkov, Sebastian Amundsen, Torbjørn V Ness, and Gaute T Einevoll. Multitask learning of biophysically-detailed neuron models. *bioRxiv*, pages 2023–12, 2023.

[125] Chaoming Wang, Yingqian Jiang, Xinyu Liu, Xiaohan Lin, Xiaolong Zou, Zilong Ji, and Si Wu. A just-in-time compilation approach for neural dynamics simulation. In *Neural Information Processing: 28th International Conference, ICONIP 2021, Sanur, Bali, Indonesia, December 8–12, 2021, Proceedings, Part III 28*, pages 15–26. Springer, 2021.

[126] Michele Migliore, C Cannia, William W Lytton, Henry Markram, and Michael L Hines. Parallel network simulations with neuron. *Journal of computational neuroscience*, 21:119–129, 2006.

[127] Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.

[128] Louis Fournier, Stéphane Rivaud, Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Can forward gradient match backpropagation? In *International Conference on Machine Learning*, pages 10249–10264. PMLR, 2023.

[129] Michael Kunst, Eva Laurell, Nouwar Mokayes, Anna Kramer, Fumi Kubo, António M Fernandes, Dominique Förster, Marco Dal Maschio, and Herwig Baier. A cellular-resolution atlas of the larval zebrafish brain. *Neuron*, 103(1):21–38, 2019.

[130] Louis K Scheffer, C Shan Xu, Michal Januszewski, Zhiyuan Lu, Shin-ya Takemura, Kenneth J Hayworth, Gary B Huang, Kazunori Shinomiya, Jeremy Maitlin-Shepard, Stuart Berg, et al. A connectome and analysis of the adult drosophila central brain. *elife*, 9, 2020.

[131] Alessandro Motta, Manuel Berning, Kevin M Boergens, Benedikt Staffler, Marcel Beining, Sahil Loomba, Philipp Hennig, Heiko Wissler, and Moritz Helmstaedter. Dense connectomic reconstruction in layer 4 of the somatosensory cortex. *Science*, 366(6469), 2019.

[132] Meike Sievers, Alessandro Motta, Martin Schmidt, Yagmur Yener, Sahil Loomba, Kun Song, Johannes Bruett, and Moritz Helmstaedter. Connectomic reconstruction of a cortical column. *bioRxiv*, pages 2024–03, 2024.

[133] Nicholas A Steinmetz, Cagatay Aydin, Anna Lebedeva, Michael Okun, Marius Pachitariu, Marius Bauza, Maxime Beau, Jai Bhagat, Claudia Böhm, Martijn Broux, et al. Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. *Science*, 372(6539), 2021.

[134] WT Lee. Tridiagonal matrices: Thomas algorithm. *MS6021, Scientific Computation, University of Limerick*, 2011.

[135] Harold S Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the ACM (JACM)*, 20(1):27–38, 1973.

[136] Boris Teodorovich Polyak. Gradient methods for minimizing functionals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 3(4):643–653, 1963.

[137] Michael Deistler, Pedro J Goncalves, and Jakob H Macke. Truncated proposals for scalable and hassle-free simulation-based inference. *Advances in Neural Information Processing Systems*, 35:23135–23149, 2022.

[138] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC, May 2011. ISBN 9780429138508.

[139] Alberto Cabezas, Adrien Corenflos, Junpeng Lao, and Rémi Louf. Blackjax: Composable Bayesian inference in JAX, 2024.

[140] Alexandros Beskos, Natesh Pillai, Gareth Roberts, Jesus-Maria Sanz-Serna, and Andrew Stuart. Optimal tuning of the hybrid monte carlo algorithm. *Bernoulli*, pages 1501–1534, 2013.

[141] Maneesh Sahani and Jennifer Linden. Evidence optimization techniques for estimating stimulus-response functions. *Advances in neural information processing systems*, 15, 2002.

[142] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[143] Kanaka Rajan and L. F. Abbott. Eigenvalue spectra of random matrices for neural networks. *Phys. Rev. Lett.*, 97:188104, Nov 2006.

[144] GV Puskorius and LA Feldkamp. Truncated backpropagation through time and kalman filter training for neurocontrol. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 4, pages 2488–2493. IEEE, 1994.

Supplementary figures

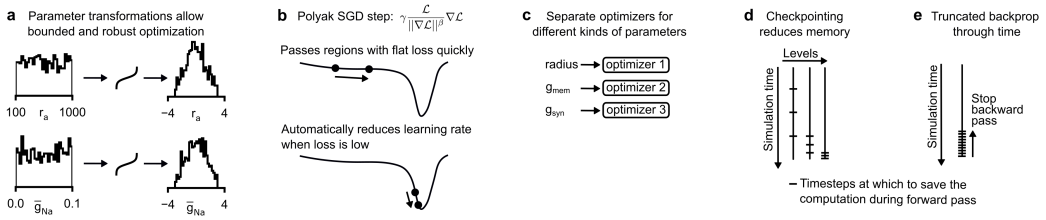


Figure S1. Robust and efficient gradient descent. (a) Histogram of two biophysical parameters (rows, random samples within optimization bounds) before (left) and after (right) parameter transformations. The transformation is designed such that the parameters are unconstrained and of the same scale. (b) Illustrative loss surface and update step made by our variant of Polyak gradient descent [56]. (c) We use different optimizers for different kinds of parameters. (d) Illustration of multi-level checkpointing [55]. We use checkpointing to overcome memory limitations of backpropagation of error. Multi-level checkpointing requires multiple forward passes per gradient computation, but typically reduces memory requirements. (e) Illustration of truncated backpropagation through time [144]. Truncated backpropagation through time allows to overcome vanishing or exploding gradients at the cost of providing only an approximate gradient.

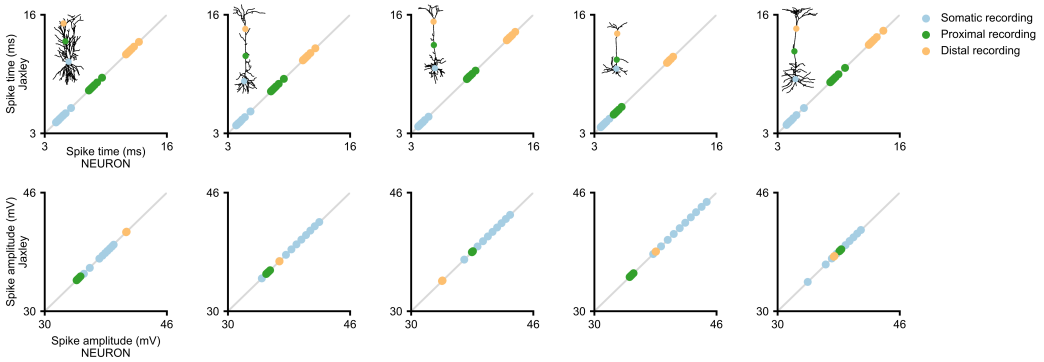


Figure S2. Accuracy of the implicit solver in JAXLEY. Top: Spike time at three recording sites (somatic, proximal, distal) for input stimuli of 10 different amplitudes, ranging from 0.2 nA to 1.1 nA (individual dots). Columns are different morphologies from the Allen Cell Types Database. Bottom: Same as top, for spike amplitude.

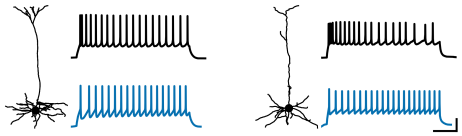


Figure S3. Gradient descent fits to recordings from the Allen Cell Types Database. Scalebars: 200 ms and 30 mV.

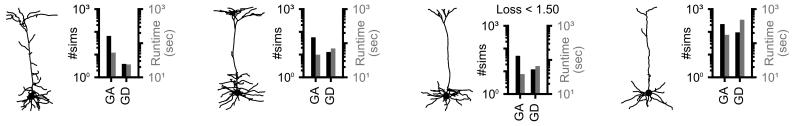


Figure S4. Fitting models to recordings from the Allen Cell Types Database. Number of simulations and runtime to reach a loss value of 1.5 or lower.

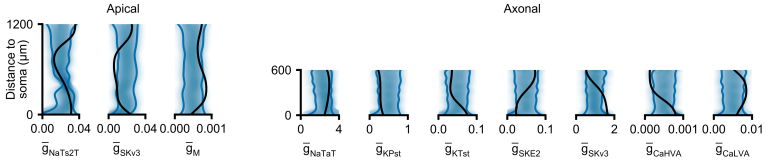


Figure S5. Bayesian inference of membrane conductances. We used JAXLEY with gradient-based Hamiltonian Monte–Carlo to infer the posterior distribution over membrane conductances of a layer 5 pyramidal cell. Blue lines are 90 % confidence intervals, black line is the ground truth that was used to generate the synthetic observation.

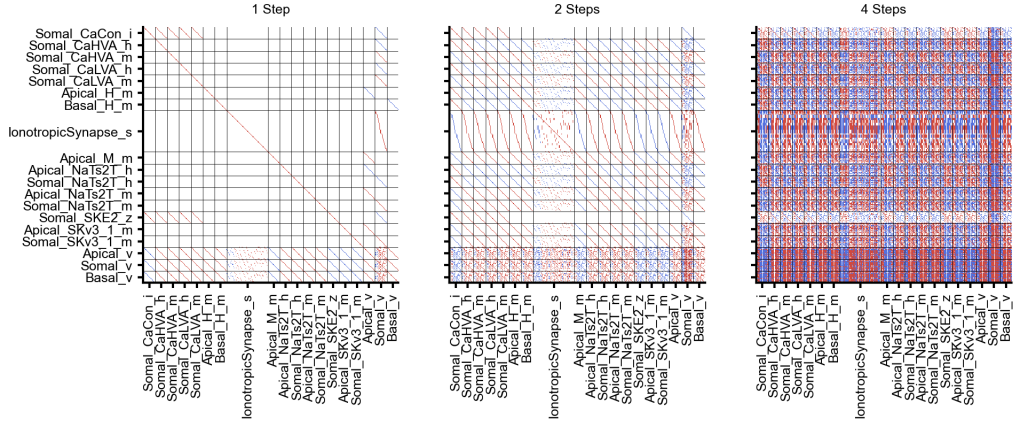


Figure S6. Jacobians reveal interactions between states JAXLEY allows us to compute Jacobians of biophysical networks by using automatic differentiation. Here we show $D\mathbf{f}|_{\mathbf{x}_0}$, where $D\mathbf{f}$ denotes the Jacobian with respect to $\mathbf{f}$, and $\mathbf{f}$ is 1, 2, and 4 steps of simulation with initial state $\mathbf{x}_0$. We used a recurrent network similar to those used in Fig. 5 (here: 20 units of which 5 are inhibitory; connection probability 0.2). As the scale between different states can be very different, we here just show the sign (red is positive, white is zero, and blue is negative).

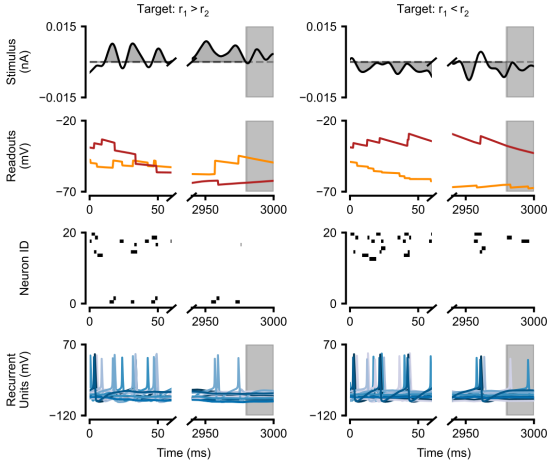


Figure S7. Generalization of the evidence integration task. Evidence integration task performance on three seconds of stimulus with positive integral (left) and negative integral (right). We used the same network parameters as in Fig. 5.

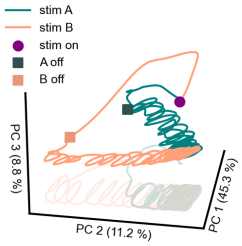


Figure S8. Long-term dynamics reveal transient coding. We here studied the long-term dynamics of the network from Fig. 5 trained to perform the delayed-match-to-sample task, in order to gain mechanistic insight into the model [101]. The figure shows the dynamics of the full state of the model (including ion concentrations), projected into principal component space, after briefly presenting either stimulus. We first smoothed each dynamic variable with a Hann window of 40 ms, and normalised, before computing the principal components. We find that, after stimulus offset, the trajectories first stayed separated, allowing the stimulus identity to be maintained for a duration that was long enough to bridge the delay period of 500 ms. Eventually, both trajectories ended on the same attractor and information about stimulus identity is lost.

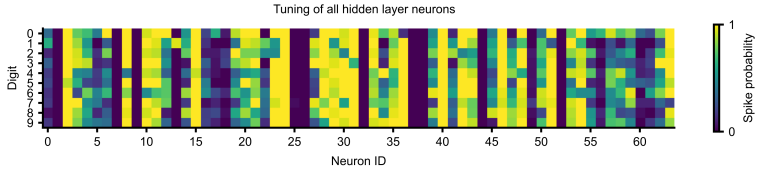


Figure S9. Hidden layer tuning of all neurons after training. We evaluated the fraction of images in the dataset for which each hidden neuron spiked.

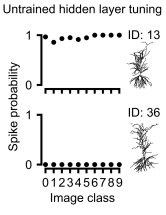


Figure S10. Hidden layer tuning before training. Before training, the two shown cells were untuned, but they developed tuning to specific digits (ON-tuning for digit '1' for ID 13 and OFF-tuning for digit '0' for ID 36) after training (Fig. 5g).

B EXTENDED AUTHOR CONTRIBUTIONS

As described in the author contribution statement at the beginning of this thesis, all text in the main part of this thesis was written by myself and all figures in the main part of this thesis were drawn or plotted by myself. The cell shown on the thesis cover is a von Economo neuron based on a reconstruction by Watson, Jones & Allman [159] and was downloaded from NeuroMorpho [133].

Below, I detail author contributions for the individual publications included with this thesis.

Energy-efficient network activity from disparate circuit parameters

This paper is co-authored by me, Jakob H. Macke, and Pedro J. Gonçalves. Pedro J. Gonçalves initially suggested to inspect the energy consumption of the pyloric network. Following the observation that the energy consumption differed largely, all authors conceived the study. As the leading author of this paper, I wrote all code, performed all experiments, and generated all figures, with input from all authors. I wrote an initial draft for the manuscript, which all authors revised and completed.

Truncated proposals for scalable and hassle-free simulation-based inference

This paper is co-authored by me, Pedro J. Gonçalves, and Jakob H. Macke. I conceived the project based on discussions with Pedro J. Gonçalves and Jakob H. Macke. I implemented the algorithm, conducted all experiments, and prepared all figures, with input from all authors. I wrote an initial draft of the paper which all authors revised.

Generalized Bayesian Inference for Scientific Simulators via Amortized Cost Estimation

This paper is co-authored by Richard Gao, me, and Jakob H. Macke. I am equally contributing lead-author with Richard Gao. Richard Gao conceived the idea to develop a method which samples parameters proportional to their distance in data space, based on discussions with Jakob H. Macke and me. Richard Gao and I then developed the methodology, based on discussions with Jakob H. Macke. Richard Gao implemented the algorithm and the benchmark, I implemented the Hodgkin–Huxley model task. Richard Gao and I wrote an initial draft of the manuscript, which all authors revised and completed.

`sbi`: A toolkit for simulation-based inference

This paper is co-authored by Álvaro Tejero-Cantero, Jan Boelts, me, Jan-Matthis Lueckmann, Conor Durkan, Pedro J. Gonçalves, David S. Greenberg, and Jakob H. Macke. I am equally contributing lead author together with Álvaro Tejero-

Cantero, Jan Boelts, Jan-Matthis Lueckmann, and Conor Durkan. This project was conceived in discussions between all authors. We contacted Conor Durkan about our plan to implement the `sbi` toolkit, and he sent us his code-base on likelihood-free inference, which we used as starting point for the project. Throughout the next weeks, Álvaro Tejero-Cantero, Jan Boelts, Jan-Matthis Lueckmann, and I contributed equally to the toolkit, with help from all authors. Álvaro Tejero-Cantero wrote an initial draft of the software paper, which all authors revised and completed.

`sbi` reloaded: A toolkit for simulation-based inference workflows

This paper is co-authored by a set of 33 authors. I am equally contributing lead author together with Jan Boelts. Together with Jan Boelts, I have maintained the `sbi` toolbox since its initial release. All other co-authors have contributed signficantly to the toolbox in different scope and on different submodules of the software, ranging from tutorials and new features to bug fixes. All contributions were coordinated by Jan Boelts and me.

Since the initial release of the toolbox, I have implemented several essential features for the toolbox. To name a few: I revised the API for multi-round inference to make it more useable, I implemented rejection and importance sampling, I developed an entirely new API for the toolbox (which is the interface that is used to this day) that allows to use pre-simulated data without access to the simulator, I implemented functionality to compute the maximum-a-posteriori estimate, I implemented automatic posterior transformation with mixture-density networks, I developed a new API for sampling methods, and I developed a new unified API for interacting with density estimators and for exposing the training loop. Together with Jan Boelts, I also invested significant effort to build a community around the `sbi` toolbox. We organized hackathons (together with Manuel Gloeckler and Guy Moss), applied (successfully) for NumFocus affiliation, and are maintaining a discord server. Together with Jan Boelts and with help from Manuel Gloeckler, I also monitor and respond to user feedback and issue reports.

I wrote an initial draft for the software paper together with Jan Boelts, which all authors revised and completed.

Differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics

This paper is co-authored by me, Kyra L. Kadhim, Matthijs Pals, Jonas Beck, Ziwei Huang, Manuel Gloeckler, Janne K. Lappalainen, Cornelius Schröder, Philipp Berens, Pedro J. Gonçalves, and Jakob H. Macke. I am the lead author. The idea to optimize biophysical models with gradient descent was conceived by Philipp Berens, Pedro J. Gonçalves, and Jakob H. Macke, and me. I developed an initial differentiable biophysics simulation in JAX based on discussions with Pedro J. Gonçalves and Jakob H. Macke. I then extended this simulation into an initial

version of the toolbox, based on discussions with Jonas Beck, Janne K. Lappalainen, Philipp Berens, Pedro J. Gonçalves, and Jakob H. Macke. Jonas Beck and Kyra L. Kadhim implemented several new features and improvements to the simulator. Ziwei Huang and Cornelius Schröder wrote a wide range of channel models for use with the simulator and provided input on how to best implement channel models.

I implemented and evaluated all benchmark tasks (Fig. 1). I implemented, evaluated, and made the figures for all single neuron tasks (Fig. 2), apart from panel 2h (Bayesian inference), which was conceived and implemented by Manuel Gloeckler. Philipp Berens conceived the task of fitting calcium measurements of retinal ganglion cell activity (RGC). I implemented, trained, and evaluated the RGC task (Fig. 3), with input from Jonas Beck, Ziwei Huang, Kyra L. Kadhim, and Philipp Berens. Kyra L. Kadhim and Matthijs Pals implemented the recurrent neural network task (Fig. 4) and prepared the figure. I implemented the MNIST task and trained initial networks (Fig. 5), Janne K. Lappalainen improved the training loop and trained the final models (Fig 5d). I prepared the figure.

All figures were revised and improved in regular meetings between all authors. I coordinated all development in this project. I wrote an initial draft of the paper, with input and changes by Philipp Berens, Pedro J. Gonçalves, and Jakob H. Macke. All authors revised and completed the manuscript.