A Differentiable Approach to Multi-scale Brain Modeling

Chaoming Wang [1] **Muyang Lyu** [2 3] **Tianqiu Zhang** [2 3] **Sichao He** [2 3] **Si Wu** [1 2 3 4 5 6]

Abstract

We present a multi-scale differentiable brain modeling workflow utilizing BrainPy [1], [2], a unique differentiable brain simulator that combines accurate brain simulation with powerful gradient-based optimization. We leverage this capability of BrainPy across different brain scales. At the single-neuron level, we implement differentiable neuron models and employ gradient methods to optimize their fit to electrophysiological data. On the network level, we incorporate connectomic data to construct biologically constrained network models. Finally, to replicate animal behavior, we train these models on cognitive tasks using gradient-based learning rules. Experiments demonstrate that our approach achieves superior performance and speed in fitting generalized leaky integrate-and-fire and Hodgkin-Huxley single neuron models. Additionally, training a biologically-informed network of excitatory and inhibitory spiking neurons on working memory tasks successfully replicates observed neural activity and synaptic weight distributions. Overall, our differentiable multi-scale simulation approach offers a promising tool to bridge neuroscience data across electrophysiological, anatomical, and behavioral scales.

1. Introduction

Modeling the entire human brain within a computer has been a long-standing dream for humanity [3]. However, it represents an immense challenge, as the accurate construction of whole-brain models that coherently link multiple spatial scales faces the obstacle of insufficient biological data collection [4]. Despite the numerous efforts dedicated to recording and measuring the brain, our observations remain partial, and the information gathered from experimental recordings falls far short of what is necessary to simulate a realistic brain [5]. For instance, at the single neuron level, neurons exhibit diverse firing patterns, while their underlying ionic channels are difficult to discern. Automatic neuron fitting has therefore become a valuable tool for bridging the gap between models and recorded neuronal data, as it can estimate the parameters of these models [6], [7]. At the network level, we have recorded neural activities such as magnetoencephalography (MEG), electroencephalography (EEG), and functional magnetic resonance imaging (fMRI) under diverse conditions. However, we still do not fully understand why the underlying neuronal circuits produce such neural activities, despite the availability of connectome data [8]–[10]. At the behavioral level, brain simulation network models still struggle to replicate the behavior of how the animal performs cognitive tasks [11]–[13].

Consequently, achieving accurate multi-scale brain modeling necessitates the development of highly efficient optimization methods capable of seamlessly integrating and reconciling data across multiple scales, spanning from individual neurons to large-scale neural networks and cognitive processes. However, conventional brain simulators, such as NEURON [14], NEST [15], and Brian2 [16], pose significant challenges for high-order optimization due to their inherent black-box nature and lack of differentiability. The absence of differentiability restricts researchers to slower and less efficient optimization techniques, and even manual heuristic parameter searches [13]. Moreover, the inability to leverage powerful gradient-based optimization techniques usually lead to longer computation times, and suboptimal model fits, further impeding the scalability of larger and more complex systems and exacerbating the challenges of multi-scale brain modeling.

To overcome these limitations, there is a pressing need for brain simulation frameworks that are natively differentiable, enabling efficient gradient-based optimizations. Recently, BrainPy [1], [2] has been proposed as a differentiable brain simulator to bridge this gap. By introducing fundamental features of a brain simulator, such as event-driven computation, sparse operators, numerical integrators, and a multi-

---

[1]School of Psychological and Cognitive Sciences, [2]Academy for Advanced Interdisciplinary Studies, [3]Peking-Tsinghua Center for Life Sciences, [4]IDG/McGovern Institute for Brain Research, [5]Center of Quantitative Biology, [6]Bejing Key Laboratory of Behavior and Mental Health, Peking University, Beijing, China. Correspondence to: Chaoming Wang <wangchaoming@pku.edu.cn>, Si Wu <siwu@pku.edu.cn>.

scale model building interface, into the numerical computing framework JAX [17], BrainPy enables faithful brain simulation while inheriting the automatic differentiation (autograd) capabilities of JAX.

Leveraging BrainPy's strengths, we propose a workflow for differentiable multi-scale brain modeling (Figure 1). This workflow utilizes gradient-based optimization to fit differentiable models of single neurons and synapses. We then incorporate connectomic data from neuroscience experiments to construct data-driven, biologically constrained spiking neural networks (SNNs). Finally, to replicate animal behavior, we train these biological-informed models on cognitive tasks using gradient-based online learning rules. Our experiments demonstrate the feasibility of our approach for achieving accurate multi-scale brain modeling.

2. Methods

We first present designs to enable our entire workflow differentiable.

2.1. Differentiable neuron models with surrogate gradients

Biological neurons generate non-differentiable binary spike events. This discontinuous nature of spiking operation, represented by the Heaviside function $\mathcal{H}(\mathbf{v})$, where $\mathbf{v}$ is the membrane potential, poses a challenge in applying gradient-based optimizations to SNN models. This is because the derivative of spiking operation is a Dirac delta function $\delta(\mathbf{v})$. In practice, surrogate gradients, which replace the delta gradient function with a smooth surrogate function, such as Gaussian [18], linear [19], SLayer [20], or multi-Gaussian function [18], have demonstrated their efficacy in training SNNs using gradient descent [21]–[23]. We apply this approach in our workflow to enable gradient-based optimization. Moreover, we provide a suite of surrogate gradient functions (listed in Appendix C) to facilitate the selection of the most suitable function for a given task.

2.2. Event-driven differentiable synaptic operators

To mimic the brain's efficient communication, traditional brain simulators leverage custom data structures for event-driven computations and spike communication [16], [24], [25]. However, these approaches often clash with autograd systems, hindering gradient-based optimization of synaptic computations. We address this challenge by introducing differentiable event-driven synaptic operators compatible with autograd frameworks (details in Appendix D). We utilize the compressed sparse row (CSR) format for storing synaptic connections and implement event-driven operations based on CSR arrays (Listing S2). Notably, these operators provide both forward and backward differentiation

rules for differentiable computations (Listing S3). Furthermore, BrainPy's event-driven operators achieve significant speedups (one to two orders of magnitude) compared to traditional sparse and dense alternatives [1], [2]. This efficiency benefit applies to both forward state computations and backward gradient calculations.

3. Workflow for multi-scale differentiable brain modeling

Based on the differentiable neuronal and synaptic building blocks described earlier, we present a workflow for multi-scale differentiable brain modeling. This approach seamlessly integrates microscopic neuron models, mesoscopic neural circuit connectivity, and macroscopic computational tasks through gradient-based optimization algorithms (see Figure 1).
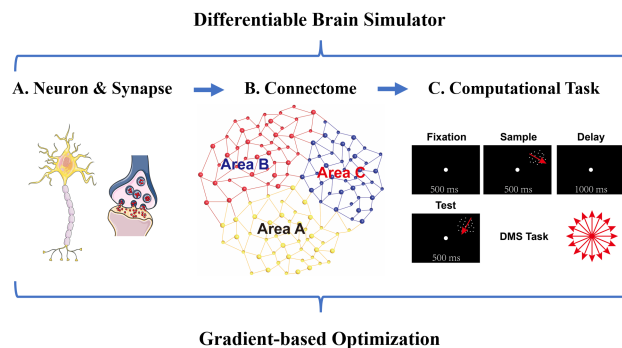

Figure 1. Multi-scale differentiable brain modeling workflow. The entire workflow is executed using the differentiable brain simulator BrainPy [1], [2]. (A) At the microscale level, the single neuron and synapse model are fitted based on electrophysiological recording data and gradient-based optimizations. (B) At the mesoscopic level, connectome constraints are incorporated into the network construction, facilitating the integration of structural connectivity information. (C) At the macroscale behavior level, gradient-based optimization methods are applied to train the above data-constrained model networks to reproduce the cognitive behaviors as seen in humans or animals.

At the single neuron and synapse level (Figure 1A), accurate modeling of individual neurons and synaptic currents is made possible by existing knowledge and experimental techniques in cellular biophysics [26], [27]. To facilitate large-scale neuronal network simulation and training, we employ point models, which capture diverse cellular behaviors while remaining differentiable and computationally efficient. Furthermore, to construct large sets of neuron models from empirical datasets conveniently and quickly, we employ gradient-based optimization (e.g., L-BFGS-B algorithm [28], [29]). This fitting procedure, powered by JAX [17], is easily scalable and parallelizable through the `jax.vmap` or `jax.pmap` semantics.

At the network level (Figure 1B), we incorporate brain structure and connectome information to construct realistic brain models. Universal function approximation [30], [31] and Kolmogorov-Arnold representation theorems [32], [33] suggest that distinct neural networks with complete different connectivity can perform the same computational tasks. Therefore, utilizing brain connectome-constrained neural models is an essential step in linking the organizational features of neuronal networks and the spectrum of cortical functions. Recent quantitative databases of the connectomes of various animals (e.g., Drosophila [10], Zebrafish [34], macaque [35], [36], mice [37], [38], and marmoset [39]), have provided rich resources for this purpose.

At the behavioral level (Figure 1C), we utilize gradient-based optimizations to train above brain data-constrained networks on computational tasks. While handcrafted tuning and manually engineered network connectivity can implement specific functions, they fall short in generating brain-scale intelligence. Here, we optimize unknown network parameters using deep learning techniques [40], enabling the model to learn and perform tasks similar to an animal. Notably, we employ the online learning method from Brain-Scale [41], which offers an online approximation of backpropagation with low computational complexity and high training performance.

4. Training biologically-informed spiking networks on cognitive tasks

To exemplify the proposed workflow, we conducted training on a biologically-informed excitatory and inhibitory (EI) spiking network using a working memory task. The dynamics of spiking neurons in our EI network are governed by generalized integrate-and-fire (GIF) neurons [42]. The synaptic dynamics are implemented using the Exponential model. To accurately capture the characteristic tonic spiking and adaptation [27], [43], the firing pattern of each GIF neuron is optimized using the L-BFGS-B algorithm (Section 4.1). The N neurons are divided into excitatory and inhibitory neurons with a 4:1 EI ratio. The connectivity between the N neurons in the network is established based on principles derived from the neocortical connectome [44]. For training both the excitatory and inhibitory weights to perform the working memory tasks (Section 4.2), we employed the online learning framework BrainScale [41]. Complete details of the EI model please see Appendix F.

4.1. Neuron fitting

Our neuron fitting procedure is depicted in Figure 2. The experimental data is obtained through current-clamp recordings, where the recorded currents mimic synaptic activity observed in *vivo* (Figure 2A). The neuron is defined in BrainPy as a differentiable model, and a loss function is

employed to quantify the disparity between the model's predictions and the experimental data. The mean square error can be used for fitting the membrane potential, while the gamma factor [45] can be employed for fitting spike trains (Appendix E). These criteria are used to calculate the gradients and subsequently update the parameter values. Through iterative gradient estimations and parameter updates, the fitting procedure aims to identify the optimal parameters that best align with the experimental recording data (Figure 2B).
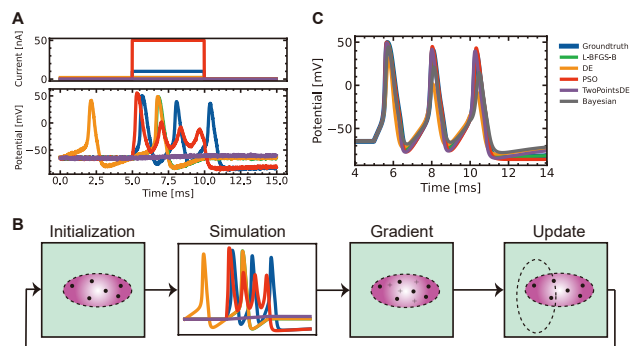

Figure 2. Overview of the neuron fitting procedure. (A) Experimental data: Step currents are injected into the neuron, and the resultant membrane potential responses are recorded. (B) Illustration of the optimization procedure: Parameter values are initialized from a distribution (initialization). Neurons with these parameters are simulated in parallel, and their outputs are compared with the ground truth data (simulation). The prediction error is utilized to estimate gradients (gradient), which are then used to update the initialized parameters for the subsequent iteration (update). (C) Fitting results of the HH model on a cortical pyramidal cell using five different optimization methods.

Our fitting method is first tested on GIF neuron models to capture characteristic cortical firing patterns such as spike frequency adaptation, phasic spiking, and rebound spiking. We compare the performance and speed of our gradient-based methods with conventional optimization algorithms, namely differential evolution (DE), DE algorithm with two points crossover (TwoPointsDE), and particle swarm optimization (PSO) provided in `Nevergrad` [46], as well as the Bayesian optimization method in `scikit-optimize` [47]. The experiments demonstrate that L-BFGS-B and Bayesian optimizations exhibit the best fitting performance (Figure S7), while DE, TwoPointsDE, and PSO methods demonstrate faster fitting speed (Table S2). Although Bayesian optimization shows good performance, it converges slowly. These results indicate that the gradient-based L-BFGS-B method provides a good tradeoff between fitting performance and speed.

We further evaluate our fitting method on Hodgkin-Huxley (HH) neuron models using realistic electrophysiological recording data. Figure 2C and Figure S8 demonstrate the

application of five fitting methods to an in *vitro* intracellular recording of a cortical pyramidal cell. The fitting results reveal that L-BFGS-B exhibits the best fitting performance (Figure 2C), achieving nearly perfect fitting of membrane potentials with a loss close to zero (Table S3). Moreover, our fitting methods demonstrate comparable speed to the evolutionary algorithm while being significantly faster than the Bayesian optimization method (Table S3). These findings underscore the potential of differentiable optimization as a promising approach for neuronal fitting.

4.2. Task training

Understanding how the brain performs complex computations remains a challenge. Recent advances in training recurrent neural networks have demonstrated high performance across various tasks, offering a promising avenue for uncovering the underlying dynamical and computational mechanisms involved [48], [49]. However, these networks often lack essential biological constraints, such as spike-based communication, structural connectivity, and the distinction between excitatory and inhibitory neurons. In this study, we propose training biological SNNs while explicitly considering electrophysiological, anatomical, and structural constraints. Specifically, we construct a foundational EI network using conductance-based GIF neurons fitted to data (see Section 4.1), incorporating connectomic connectivity [44] and conductance-based synaptic dynamics [50] to implement the working memory task through gradient-based optimization algorithms [41].
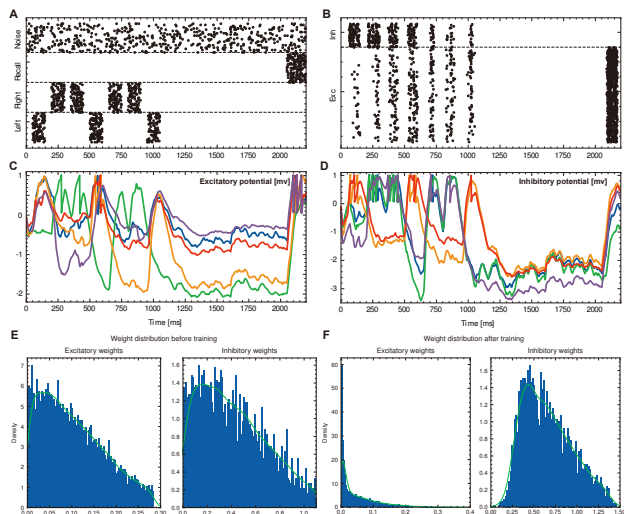

Figure 3. Training the biological-informed excitatory and inhibitory spiking networks using the evidence accumulation task. (A) The input spike train. (B) The recurrent spiking dynamics. (C, D) The membrane potentials of five excitatory (C) and inhibitory (D) neurons. (E, F) The synaptic weight distribution before (E) and after (F) training.

To generate the training data, we followed the experimental

setup of an evidence accumulation task [51]. The input spike train was divided into four segments: the left and right stimuli, the recall cue, and the background noise (Figure 3A). Our network was trained to separately count the left and right cues and generate the correct response by comparing the resulting numbers after prolonged periods of delay. We recorded the responses of both the excitatory and inhibitory neurons in the recurrent layer (Figure 3B). During the evidence accumulation period, inhibitory neurons exhibited significant responses after each stimulus presentation, whereas excitatory neurons displayed lower firing rates. However, during the recall period, inhibitory neurons rarely spiked. We also examined the membrane potential of all neurons (Figure 3C and D). In contrast to the current-based synapse model commonly used in deep learning applications, our conductance-based synapse modeling ensured that the membrane potential remained constrained between excitatory and inhibitory reversal potentials, eliminating the need for voltage regularization. Additionally, we analyzed the synaptic weight distribution before and after training (Figure 3E and F). We initialized excitatory and inhibitory weights with a normal distribution and took their absolute values (Figure 3E). After training, synaptic weights exhibited a distribution similar to that observed in biological measurements. Specifically, excitatory weights followed the tail of a Gaussian distribution [52], while inhibitory weights showed a log-normal distribution [53], [54].

5. Conclusion and discussion

We proposed a novel workflow for differentiable multi-scale brain modeling by integrating various levels of information and constraints to build brain models that can reproduce cognitive behaviors observed in humans or animals. We demonstrated this workflow by training a biologically informed GIF network to accomplish an evidence accumulation task. Although the current illustration utilizes a network with hundreds of neurons, the online learning algorithms employed are readily scalable to much larger models (refer to Appendix G for scalability analysis). Overall, our proposed differentiable approach has the potential to accelerate progress in developing accurate and biologically plausible multi-scale brain models, ultimately leading to a deeper understanding of the brain.

However, many important challenges remain to be addressed in the future (see Appendix H for details). These challenges include balancing data quality and availability with model realism, determining the appropriate granularity for simplifying and approximating biological processes within the model, and ensuring the interpretability and theoretical grounding of the derived models. Additionally, addressing the computational efficiency of handling large-scale networks with high-dimensional parameter spaces is crucial.

References

[1] C. Wang, T. Zhang, X. Chen, S. He, S. Li, and S. Wu, "Brainpy, a flexible, integrative, efficient, and extensible framework for general-purpose brain dynamics programming," *Elife*, vol. 12, e86365, 2023.

[2] C. Wang, T. Zhang, H. Gu, S. He, S. Li, and S. Wu, "A differentiable brain simulator bridging brain simulation and brain-inspired computing," in *The Twelfth International Conference on Learning Representations*, 2024.

[3] K. Amunts, M. Axer, S. Banerjee, *et al.*, "The coming decade of digital brain research: A vision for neuroscience at the intersection of technology and computing," *Imaging Neuroscience*, vol. 2, pp. 1–35, 2024.

[4] E. D'Angelo and V. Jirsa, "The quest for multiscale brain modeling," *Trends in neurosciences*, vol. 45, no. 10, pp. 777–790, 2022.

[5] L. K. Scheffer and I. A. Meinertzhagen, "A connectome is not enough–what is still needed to understand the brain of drosophila?" *Journal of Experimental Biology*, vol. 224, no. 21, jeb242740, 2021.

[6] C. Rossant, D. F. Goodman, B. Fontaine, J. Platkiewicz, A. K. Magnusson, and R. Brette, "Fitting neuron models to spike trains," *Frontiers in neuroscience*, vol. 5, p. 8742, 2011.

[7] C. Rossant, D. F. Goodman, J. Platkiewicz, and R. Brette, "Automatic fitting of spiking neuron models to electrophysiological recordings," *Frontiers in neuroinformatics*, vol. 4, p. 1273, 2010.

[8] P. K. Shiu, G. R. Sterne, N. Spiller, *et al.*, "A leaky integrate-and-fire computational model based on the connectome of the entire adult drosophila brain reveals insights into sensorimotor processing," *bioRxiv*, 2023.

[9] S. Dorkenwald, A. Matsliah, A. R. Sterling, *et al.*, "Neuronal wiring diagram of an adult brain," *bioRxiv*, 2023.

[10] M. Winding, B. D. Pedigo, C. L. Barnes, *et al.*, "The connectome of an insect brain," *Science*, vol. 379, no. 6636, eadd9330, 2023.

[11] T. C. Potjans and M. Diesmann, "The cell-type specific cortical microcircuit: Relating structure and activity in a full-scale spiking network model," *Cerebral cortex*, vol. 24, no. 3, pp. 785–806, 2014.

[12] M. Schmidt, R. Bakker, K. Shen, G. Bezgin, M. Diesmann, and S. J. van Albada, "A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas," *PLOS Computational Biology*, vol. 14, no. 10, e1006359, 2018.

[13] Y. N. Billeh, B. Cai, S. L. Gratiy, *et al.*, "Systematic integration of structural and functional data into multi-scale models of mouse primary visual cortex," *Neuron*, vol. 106, no. 3, pp. 388–403, 2020.

[14] O. Awile, P. Kumbhar, N. Cornu, *et al.*, "Modernizing the neuron simulator for sustainability, portability, and performance," *Frontiers in Neuroinformatics*, vol. 16, p. 884 046, 2022.

[15] M.-O. Gewaltig and M. Diesmann, "Nest (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[16] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *elife*, vol. 8, e47314, 2019.

[17] R. Frostig, M. J. Johnson, and C. Leary, "Compiling machine learning programs via high-level tracing," *Systems for Machine Learning*, vol. 4, no. 9, 2018.

[18] B. Yin, F. Corradi, and S. M. Bohté, "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks," *Nature Machine Intelligence*, vol. 3, no. 10, pp. 905–913, 2021.

[19] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," *Advances in neural information processing systems*, vol. 31, 2018.

[20] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," *Advances in neural information processing systems*, vol. 31, 2018.

[21] S. M. Bohte, "Error-backpropagation in networks of fractionally predictive spiking neurons," in *International conference on artificial neural networks*, Springer, 2011, pp. 60–68.

[22] E. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, pp. 51–63, Nov. 2019. DOI: 10.1109/MSP.2019.2931595.

[23] G. Bellec, F. Scherr, A. Subramoney, *et al.*, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature communications*, vol. 11, no. 1, p. 3625, 2020.

[24] S. Kunkel, T. C. Potjans, J. M. Eppler, H. E. Plesser, A. Morrison, and M. Diesmann, "Meeting the memory challenges of brain-scale network simulation," *Frontiers in neuroinformatics*, vol. 5, p. 35, 2012.

[25] S. Kunkel, M. Schmidt, J. M. Eppler, *et al.*, "Spiking network simulation code for petascale computers," *Frontiers in neuroinformatics*, vol. 8, p. 78, 2014.

[26] C. Koch and I. Segev, *Methods in neuronal modeling: from ions to networks*. MIT press, 1998.

[27] C. Teeter, R. Iyer, V. Menon, *et al.*, "Generalized leaky integrate-and-fire models classify multiple neuron types," *Nature communications*, vol. 9, no. 1, p. 709, 2018.

[28] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[29] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on scientific computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[30] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[31] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.

[32] A. N. Kolmogorov, *On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables*. American Mathematical Society, 1961.

[33] J. Braun and M. Griebel, "On a constructive proof of kolmogorov's superposition theorem," *Constructive approximation*, vol. 30, pp. 653–675, 2009.

[34] M. Kunst, E. Laurell, N. Mokayes, *et al.*, "A cellular-resolution atlas of the larval zebrafish brain," *Neuron*, vol. 103, no. 1, pp. 21–38, 2019.

[35] N. T. Markov, M. M. Ercsey-Ravasz, A. Ribeiro Gomes, *et al.*, "A weighted and directed interareal connectivity matrix for macaque cerebral cortex," *Cerebral cortex*, vol. 24, no. 1, pp. 17–36, 2014.

[36] N. T. Markov, J. Vezoli, P. Chameau, *et al.*, "Anatomy of hierarchy: Feedforward and feedback pathways in macaque visual cortex," *Journal of comparative neurology*, vol. 522, no. 1, pp. 225–259, 2014.

[37] S. W. Oh, J. A. Harris, L. Ng, *et al.*, "A mesoscale connectome of the mouse brain," *Nature*, vol. 508, no. 7495, pp. 207–214, 2014.

[38] B. Zingg, H. Hintiryan, L. Gou, *et al.*, "Neural networks of the mouse neocortex," *Cell*, vol. 156, no. 5, pp. 1096–1111, 2014.

[39] P. Majka, S. Bai, S. Bakola, *et al.*, "Open access resource for cellular-resolution analyses of corticocortical connectivity in the marmoset monkey," *Nature communications*, vol. 11, no. 1, p. 1133, 2020.

[40] A. Saxe, S. Nelli, and C. Summerfield, "If deep learning is the answer, what is the question?" *Nature Reviews Neuroscience*, vol. 22, no. 1, pp. 55–67, 2021.

[41] C. Wang, X. Dong, J. Jiang, and S. Wu, "*BrainScale*: Enabling scalable online learning in spiking neural networks," *Submitted*, 2024.

[42] R. Jolivet, T. J. Lewis, and W. Gerstner, "Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy," *Journal of neurophysiology*, vol. 92, no. 2, pp. 959–976, 2004.

[43] H. Staining, "Allen cell types database," *Allen Inst. Brain Sci*, vol. 88, no. 1-9, p. 271, 2015.

[44] P. Theodoni, P. Majka, D. H. Reser, D. K. Wójcik, M. G. Rosa, and X.-J. Wang, "Structural attributes and principles of the neocortical connectome in the marmoset monkey," *Cerebral Cortex*, vol. 32, no. 1, pp. 15–28, 2022.

[45] R. Jolivet, R. Kobayashi, A. Rauch, R. Naud, S. Shinomoto, and W. Gerstner, "A benchmark test for a quantitative assessment of simple neuron models," *Journal of neuroscience methods*, vol. 169, no. 2, pp. 417–424, 2008.

[46] P. Bennet, C. Doerr, A. Moreau, J. Rapin, F. Teytaud, and O. Teytaud, "Nevergrad: Black-box optimization platform," *ACM SIGEVOlution*, vol. 14, no. 1, pp. 8–15, 2021.

[47] G. Louppe, "Bayesian optimisation with scikit-optimize," in *PyData Amsterdam*, 2017.

[48] H. F. Song, G. R. Yang, and X.-J. Wang, "Training excitatory-inhibitory recurrent neural networks for cognitive tasks: A simple and flexible framework," *PLoS computational biology*, vol. 12, no. 2, e1004792, 2016.

[49] O. Barak, "Recurrent neural networks as versatile tools of neuroscience research," *Current opinion in neurobiology*, vol. 46, pp. 1–6, 2017.

[50] T. P. Vogels and L. F. Abbott, "Signal propagation and logic gating in networks of integrate-and-fire neurons," *Journal of neuroscience*, vol. 25, no. 46, pp. 10 786–10 795, 2005.

[51] A. S. Morcos and C. D. Harvey, "History-dependent variability in population dynamics during evidence accumulation in cortex," *Nature neuroscience*, vol. 19, no. 12, pp. 1672–1681, 2016.

[52] B. Barbour, N. Brunel, V. Hakim, and J.-P. Nadal, "What can we learn from synaptic weight distributions?" *TRENDS in Neurosciences*, vol. 30, no. 12, pp. 622–629, 2007.

[53] Y. Loewenstein, A. Kuras, and S. Rumpel, "Multiplicative dynamics underlie the emergence of the log-normal distribution of spine sizes in the neocortex in vivo," *Journal of Neuroscience*, vol. 31, no. 26, pp. 9481–9488, 2011.

[54] G. Buzsáki and K. Mizuseki, "The log-dynamic brain: How skewed distributions affect network operations," *Nature Reviews Neuroscience*, vol. 15, no. 4, pp. 264–278, 2014.

[55] A. Teska, M. Stimberg, and R. Brette, "Brian2modelfitting," *Zenodo*, 2020.

[56] S. K. Esser, P. A. Merolla, J. V. Arthur, *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, pp. 11 441–11 446, 2016.

[57] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in Neuroscience*, vol. 12, 2018.

[58] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015.

[59] B. Yin, F. Corradi, and S. M. Bohté, "Effective and efficient computation with multiple-timescale spiking recurrent neural networks," in *International Conference on Neuromorphic Systems 2020*, ser. ICONS 2020, New York, NY, USA: Association for Computing Machinery, 2020, ISBN: 9781450388511. DOI: 10.1145/3407197.3407225. [Online]. Available: https://doi.org/10.1145/3407197.3407225.

[60] S. Yin, S. K. Venkataramanaiah, G. K. Chen, *et al.*, "Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations," in *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, IEEE, 2017, pp. 1–5.

[61] K. Suetake, S.-i. Ikegawa, R. Saiin, and Y. Sawada, S^3nn: *Time step reduction of spiking surrogate gradients for training energy efficient single-step spiking neural networks*, 2023. arXiv: 2201.10879 [cs.LG].

[62] L. Herranz-Celotti and J. Rouat, *Surrogate gradients design*, 2022. arXiv: 2202.00282 [cs.NE].

[63] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave gaussian quantization," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, IEEE Computer Society, 2017, pp. 5406–5414. DOI: 10.1109/CVPR.2017.574. [Online]. Available: https://doi.org/10.1109/CVPR.2017.574.

[64] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[65] A. Amir, B. Taba, D. Berg, *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[66] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, pp. 339–356, 1988. [Online]. Available: https://api.semanticscholar.org/CorpusID:205118721.

[67] M. C. Mozer, "A focused backpropagation algorithm for temporal pattern recognition," *Complex Syst.*, vol. 3, 1989. [Online]. Available: https : / / api . semanticscholar.org/CorpusID:18036435.

[68] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computation*, vol. 2, pp. 490–501, 1990. [Online]. Available: https://api.semanticscholar. org/CorpusID:12979634.

A. Software and Data

The in *vitro* intracellular recording of a cortical pyramidal cell can be obtained in `brain2modelfitting` [55]. BrainPy is available publicly on GitHub at `https://github.com/brainpy/BrainPy`. As of now, BrainScale [41] is undergoing a review process and is temporarily unavailable. However, it is expected to be released in the future. Additional packages related to the BrainPy ecosystem are accessible through the BrainPy GitHub organization at `https://github.com/brainpy`. The code necessary to reproduce the results presented in this paper can be found in the following GitHub repository: `https://github.com/chaoming0625/differentiable-brain-modeling-workflow`.

B. Acknowledgements

C. Surrogate gradient functions

In recent years, spiking neural networks (SNNs) have garnered attention due to their promising advantages in energy efficiency, fault tolerance, and biological plausibility. However, training SNNs using standard gradient descent methods is challenging because their activation functions are discontinuous and have near-zero gradients across most points. To tackle this issue, a common approach is to replace the non-differentiable spiking function with a surrogate gradient function [22]. A surrogate gradient function is a smooth approximation of the derivative of the activation function, enabling the application of gradient-based learning algorithms to SNNs. The BrainPy library offers a variety of surrogate gradient functions, each possessing different characteristics such as smoothness, boundedness, and biological plausibility. A comprehensive list of these functions is provided in Table S1, and an example of a surrogate gradient function is illustrated in Figure S4.

In practical applications, users can employ these surrogate gradient functions to determine whether a spike is generated at the current time step, as demonstrated in the following Python code:

```python
# V: the membrane potential
# V_th: the threshold of the membrane potential to generate a spike
spike = brainpy.math.surrogate.arctan(V - V_th)
```

Listing S1. The example code to employ the surrogate function as the method to determine whether a spike is generated.

By utilizing the appropriate surrogate gradient function, the code above allows users to assess whether a spike occurs based on the comparison between the membrane potential (`V`) and the threshold (`V_th`).

D. Event-driven synaptic operators

Synaptic computation usually needs event-driven matrix-vector multiplication $\mathbf{y} = \mathbf{Mv}$, where $\mathbf{v}$ is the presynaptic spikes, $\mathbf{M}$ the synaptic connection matrix, and $\mathbf{y}$ the postsynaptic current. Specifically, it performs matrix-vector multiplication in a sparse and efficient way by exploiting the event property of the input vector $\mathbf{v}$. Instead of multiplying the entire matrix $\mathbf{M}$ by the vector $\mathbf{v}$, which can be wasteful if $\mathbf{v}$ has many zero elements, event-driven matrix-vector multiplication in BrainPy only performs multiplications for the non-zero elements of the vector, which are called events. This can reduce the number of operations and memory accesses, and improve the running performance of matrix-vector multiplication.

Particularly, we implement event-driven operators based on arrays with the compressed sparse row (CSR) format and provide both forward and backward differentiation rules. The CSR format represents the synaptic connectivity between pre- and post-synaptic neuron populations, comprising three arrays: `<val, col_ind, row_ptr>`. `val` stores the non-zero synaptic weights, `col_ind` stores the postsynaptic indices of the corresponding non-zero weights, and `row_ptr` stores the starting indices of each presynaptic neuron in the `val` and `col_ind` arrays.

To perform an event-driven linear transformation $\mathbf{y} = \mathbf{xW}$, where $\mathbf{W}$ is the CSR-formatted connectivity, the pseudo-code is implemented as:

```python
def csrmv(val, col_ind, row_ptr, x, y):
  for i, event in enumerate(x):
    if event:
      for j in range(row_ptr[i],row_ptr[i+1]):
```
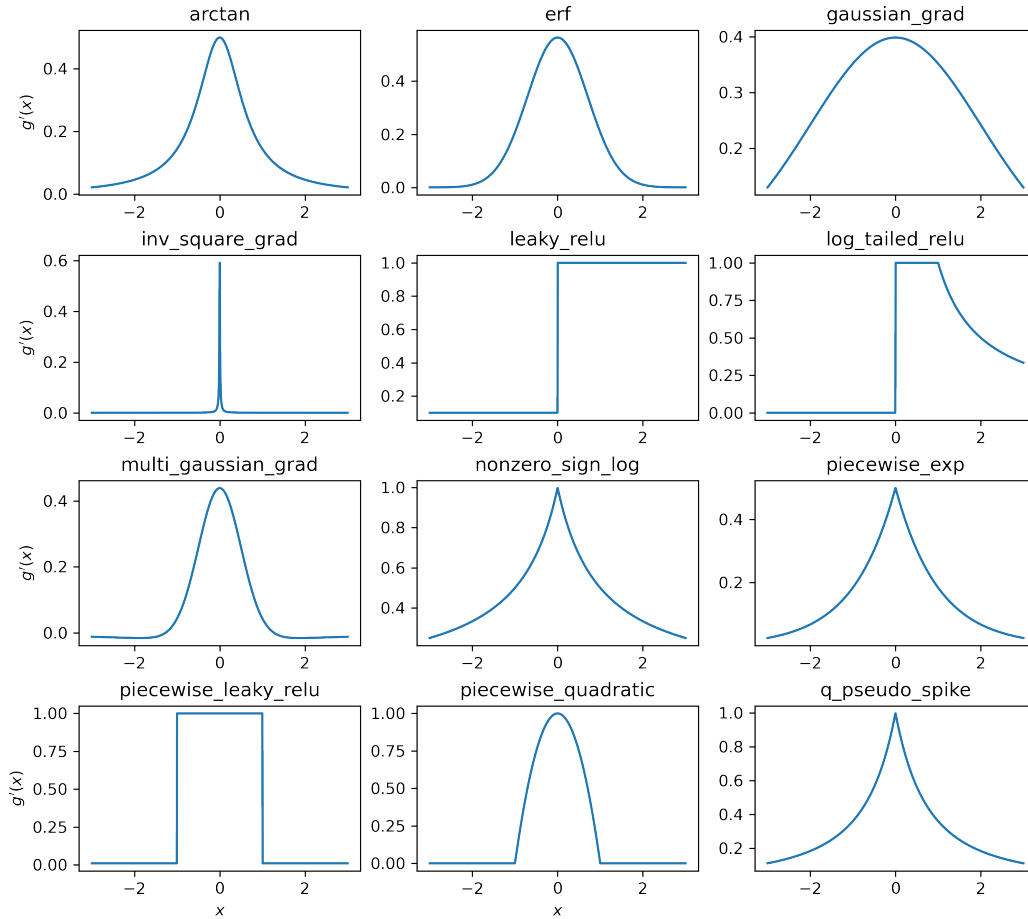
Figure S4. The collection of surrogate gradient functions $g'(x)$ in BrainPy [1], [2], where $x \geq 0$ represents the neuronal membrane potential exceeding the spiking threshold.

```
5           y[col_ind[i]] += val[j]
```

Listing S2. The forward pass of event-driven sparse matrix-vector multiplication.

To efficiently compute the gradients of $\mathrm{d}\mathbf{x}$ and $\mathrm{d}\mathbf{W}$, we implement the event-driven gradient computation as follows:

```
1  # compute dx
2  def csrmv_dx(val, col_ind, row_ptr, dy, dx):
3    for i in range(dy.shape[0]):
4      r = 0.
5      for j in range(row_ptr[i], row_ptr[i+1]):
6        r += val[j] * dy[col_ind[j]]
7      dx[i] = r
8
9  # compute dW
10 def csrmv_dW(col_ind, row_ptr, x, dy, dW):
11   for i, event in enumerate(x):
12     if event:
13       for j in range(row_ptr[i], row_ptr[i+1]):
14         dW[j] = dy[col_ind[j]]
```

Listing S3. The backward pass of event-driven sparse matrix-vector multiplication.

9

Table S1. The full list of surrogate gradient functions provided in BrainPy.

Surrogate Gradient Function	Implementation
Sigmoid function	`brainpy.math.surrogate.sigmoid`
Piecewise quadratic function [56], [57]	`brainpy.math.surrogate.piecewise_quadratic`
Piecewise exponential function [22]	`brainpy.math.surrogate.piecewise_exp`
Soft sign function	`brainpy.math.surrogate.soft_sign`
Arctan function	`brainpy.math.surrogate.arctan`
Nonzero sign log function	`brainpy.math.surrogate.nonzero_sign_log`
ERF function [57]–[59]	`brainpy.math.surrogate.erf`
Piecewise leaky ReLU function [57], [60]	`brainpy.math.surrogate.piecewise_leaky_relu`
Squarewave Fourier series	`brainpy.math.surrogate.squarewave_fourier_series`
S2NN function [61]	`brainpy.math.surrogate.s2nn`
q-PseudoSpike function [62]	`brainpy.math.surrogate.q_pseudo_spike`
Leaky ReLU function	`brainpy.math.surrogate.leaky_relu`
Log-tailed ReLU function [63]	`brainpy.math.surrogate.log_tailed_relu`
ReLU function [22]	`brainpy.math.surrogate.relu_grad`
Gaussian function [18]	`brainpy.math.surrogate.gaussian_grad`
Multi-Gaussian function [18]	`brainpy.math.surrogate.multi_gaussian_grad`
Inverse-square function	`brainpy.math.surrogate.inv_square_grad`
SLayer function [20]	`brainpy.math.surrogate.slayer_grad`

E. Loss functions for neuron fitting

E.1. Mean square error for fitting membrane potentials

In order to align the simulated membrane potential with the experimentally recorded potentials, we compute the mean squared difference between the data $\hat{Y}_i$ and the simulated trace Y_i using the mean square error formula:

$$\text{MSE} = \frac{1}{T} \sum_{i=1}^{T} (Y_i - \hat{Y}_i)^2, \tag{1}$$

where T is the total number of times.

E.2. Gamma factor for fitting spike trains

The Gamma factor [45] serves as a metric for assessing the agreement between spike timings in the simulated and target traces. It is commonly employed to evaluate the performance of spiking neuron models when fitting them to electrophysiological recordings of individual neurons. The gamma factor primarily focuses on the proportion of predicted spikes that coincide with the spikes in the recording. Essentially, it quantifies how accurately the model reproduces the timing of the neuron's firing events. The calculation of the gamma factor is as follows:

$$\Gamma = \left(\frac{2}{1 - 2\Delta r_{\text{exp}}} \right) \left(\frac{N_{\text{coinc}} - 2\delta N_{\text{exp}} r_{\text{exp}}}{N_{\text{exp}} + N_{\text{model}}} \right) \tag{2}$$

where

- N_{coinc}: number of coincidences

- N_{exp} and N_{model}: number of spikes in experimental and model spike trains

- r_{exp}: average firing rate in experimental train

- $2\Delta N_{\text{exp}} r_{\text{exp}}$: expected number of coincidences with a Poisson process

The gamma factor Γ equals 1 when the two spike trains match perfectly and decreases for less precise matches. It reaches 0 when the number of coincidences matches the expected count from two homogeneous Poisson processes with the same firing rate.

To turn the Gamma factor into a loss function, we add a correction term:

$$\text{Loss} = 1 + 2\frac{|r_{\text{data}} - r_{\text{model}}|}{r_{\text{data}}} - \Gamma, \tag{3}$$

where r_{data} and r_{model} are the firing rates measured in the data and model, respectively.

F. Excitatory and inhibitory spiking network models

F.1. Network structure

The architecture of recurrent excitatory and inhibitory spiking networks used here is shown in Figure S5, where the recurrent layer consists of excitatory and inhibitory spiking units that receive and process the time-varying inputs from the input layer, and generate the desired time-varying outputs. The input layer encodes the sensory information relevant to the task, while the readout layer produces a decision in terms of an abstract decision variable.
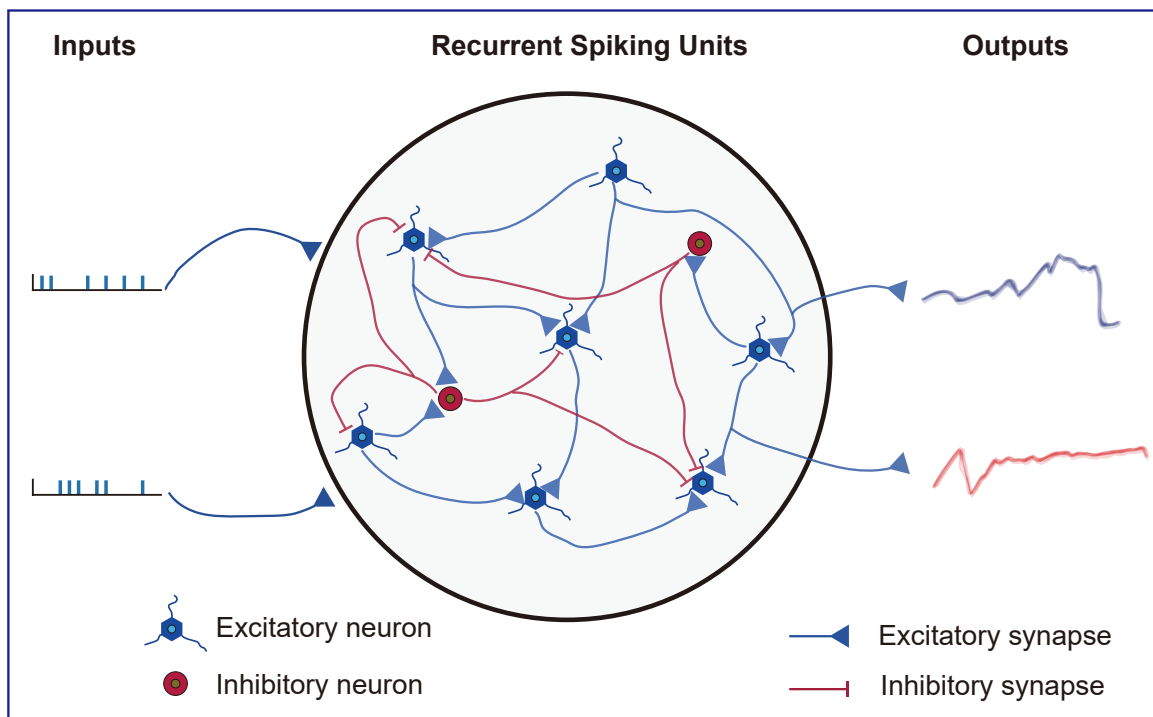

Figure S5. Architecture of the recurrent spiking EI network. The network consists of excitatory (E) and inhibitory (I) spiking units, denoted by $\mathbf{r}(t)$. These units are trained using an online gradient-based learning framework BrainScale [41]. Time-varying inputs $\mathbf{u}(t)$ are received by the network, and the recurrent activity is encoded through time-varying outputs $\mathbf{z}(t)$. The inputs represent task-relevant sensory information or internal rules, while the outputs encode a decision in the form of an abstract decision variable, probability distribution, or direct motor output. Each spiking unit exhibits its own dynamics, and the firing rate of each unit is adjusted through our differentiable fitting method (Section 4.1). The connectivity between the spiking units is determined based on connectomic measurements [44].

F.2. Input layer

The input layer in our spiking neural network is designed for an evidence accumulation task and comprises $N_{\text{in}} = 100$ spiking neurons (Figure 3A). These neurons are divided into four functionally distinct groups:

- **Left and Right Stimuli**: The first two groups represent the left and right stimuli, respectively. Each group contains 25

neurons and fires at a maximum rate of 40 Hz during the evidence accumulation period (first 1100 ms). This neural activity encodes the sensory evidence for each side.

- **Recall Signals**: The third group consists of 25 neurons that generate recall signals during the output period (last 150 ms). These neurons are crucial for retrieving relevant information from memory to aid in the decision-making process.

- **Background Noise**: The fourth group comprises 25 neurons that fire at a constant rate of 10 Hz throughout the entire simulation. This group simulates background activity from other cortical areas, adding a layer of realism to the network.

F.3. Recurrent spiking units

For the spiking model, the recurrent cell was implemented with the spiking neuron model. In this study, the spiking neuron is modified from the generalized integrate-and-fire neuron model [42]. In particular, this model has two internal currents, one fast and one slow. Its dynamic behavior is given by

$$\tau_{I1}\frac{d\mathbf{I_1}}{dt} = -\mathbf{I_1}, \qquad\qquad \text{fast internal current} \qquad\qquad (4)$$

$$\tau_{I2}\frac{d\mathbf{I_2}}{dt} = -\mathbf{I_2}, \qquad\qquad \text{slow internal current} \qquad\qquad (5)$$

$$\tau_{V}\frac{d\mathbf{V}}{dt} = -\mathbf{V} + V_{\text{rest}} + R(\mathbf{I_1} + \mathbf{I_2} + \mathbf{I}_{\text{ext}}), \qquad\qquad \text{membrane potential} \qquad\qquad (6)$$

When V^i of i-th neuron meets V_{th}, the modified GIF model fires:

$$I_1^i \leftarrow A_1, \qquad\qquad (7)$$

$$I_2^i \leftarrow I_2^i + A_2, \qquad\qquad (8)$$

$$V^i \leftarrow V_{\text{rest}}, \qquad\qquad (9)$$

where τ_{I1} denotes the time constant of the fast internal current, τ_{I2} the time constant of the slow internal current, τ_V the time constant of membrane potential, R the resistance, $\mathbf{I}_{\text{ext}}$ the external input, V_{rest} the resting potential, and A_1 and A_2 the spike-triggered currents.

To match the firing patterns observed in electrophysiological experiments, particularly the tonic spiking and adaptation, we fit the neuron parameters $A_1, A_2, \tau_{I_1}, \tau_{I_2}$ using our gradient-based optimization methods (Section 4.1).

For the forward spiking operation, we use the Heaviside function to generate the spike:

$$\text{spike}(\mathbf{x}) = \mathcal{H}(\mathbf{V}[t] - V_{th}) = \mathcal{H}(\mathbf{x}), \qquad\qquad (10)$$

where $\mathbf{x}$ is used to represent $\mathbf{V}[t] - V_{th}$.

To make the non-differentiable spiking activation compatible with the gradient-based algorithm, we considered a surrogate gradient (Appendix C):

$$\text{spike}'(\mathbf{x}) = \text{ReLU}(\alpha * (\text{width} - |\mathbf{x}|)) \qquad\qquad (11)$$

where $\text{width} = 1.0$, and $\alpha = 0.3$. α is the parameter that controls the altitude of the gradient, and width is the parameter that controls the width of the gradient.

F.4. Synapse dynamics

For modeling the synaptic connections between these neurons, we employed a conductance-based current approach. The input current (I_{ext}^i) for each neuron i is calculated as

$$I_{\text{ext}}^i = g_{\text{exc}}^i(E_{\text{exc}} - V^i) + g_{\text{inh}}^i(E_{\text{inh}} - V^i), \qquad\qquad (12)$$

where the reversal potentials are $E_{\text{exc}} = 0$ mV and $E_{\text{inh}} = -120$ mV.

The synaptic dynamics are characterized by exponential synapses,

$$\tau_{\text{syn}} \frac{d\mathbf{g}_{\text{exe/inh}}}{dt} = -\mathbf{g}_{\text{exe/inh}} \tag{13}$$

where τ_{syn} is the time constant of the synaptic state decay, and t_i^k is the k-th spiking time of the presynaptic neuron i. Moreover, the appropriate synaptic variable of the postsynaptic conductance $g_{\text{exe/inh}}^j$ increases when a presynaptic neuron (i) fires. For an excitatory presynaptic neuron,

$$g_{\text{exc}}^j \rightarrow g_{\text{exc}}^j + W_{ij}^{\text{exc}}, \tag{14}$$

and for an inhibitory presynaptic neuron,

$$g_{\text{inh}}^j \rightarrow g_{\text{inh}}^j + W_{ij}^{\text{inh}}. \tag{15}$$

Typically, we set $\tau_{\text{syn}} = 10$ ms.

F.5. Scaled membrane potential

The subthreshold dynamics of biological neurons and synapses usually operate with very negative membrane potentials, often in the range of -50 mV to -80 mV. These large negative values can create challenges during training, especially when using low-precision computing. To address this, we employ a rescaling approach that normalizes the membrane potential.

The rescaling process involves an offset value (V_{offset}) and a scaling factor (V_{scale}). Every membrane potential (V) is transformed using the following equation:

$$V_s = \frac{V - V_{\text{offset}}}{V_{\text{scale}}} \tag{16}$$

Here, V_s represents the rescaled membrane potential. By setting V_{scale} such that the firing threshold becomes 1 after rescaling, we achieve a more manageable range of values for training purposes, particularly with limited computational precision. Particularly, we use $V_{\text{scale}} = 20$ and $V_{\text{offset}} = 60$ mv, so that the membrane threshold is normalized to 1, and reversal potentials of excitatory and inhibitory synapses are rescaled to 3.0 and -3.0.

F.6. Network connectivity

We designed a network comprising 400 cells, where the excitatory to inhibitory neuron ratio was set to 4:1 (Figure S5). To establish connectivity within the network, we randomly interconnected these neurons with a connection probability of 10%. This choice of connection probability takes into account the observed higher values for neighboring neurons in the cortex, as well as the lower values for neurons that are more distant [44].

Apart from the recurrent connections, the input neurons project excitatory synapses to all the recurrent neurons. As for the readout layer, since there is no known biological correspondence for how the brain interprets the recurrent signals, we opted for a linear projection with leaky dynamics.

F.7. Readout layer

The spiking activity of the recurrent neurons is read out using a linear layer that incorporates the leaky dynamics of the neurons:

$$\tau_{\text{out}} \frac{d\mathbf{y}}{dt} = -\mathbf{y} + W^{\text{out}}\mathbf{z} + b^{\text{out}}, \tag{17}$$

where τ_{out} is the time constant of the output neuron, W^{out} the synaptic weights between the recurrent and output neurons, and b^{out} the bias. In the discrete description, the output dynamics is written as:

$$\mathbf{y}[t + \Delta t] = \alpha_{\text{out}}\mathbf{y}[t] + (W^{\text{out}}\mathbf{z}[t] + b^{\text{out}})\Delta t, \tag{18}$$

where $\alpha_{\text{out}} = e^{-\frac{1}{\tau_{\text{out}}}\Delta t}$.

F.8. Weight initialization

Initial input and recurrent weights were drawn from a Gaussian distribution and taken the absolute values $W_{ji} \sim \left| \sqrt{\frac{s}{n_{\text{in}}}} \mathcal{N}(0,1) \right|$, where n_{in} is the number of afferent neurons, $\mathcal{N}(0,1)$ is the zero-mean unit-variance Gaussian distribution, and s is the weight scale. For excitatory neurons (including the input and recurrent excitatory neurons), $s = 1.0$; for inhibitory neurons, $s = 4.0$. For the readout weights, we draw its values from a Gaussian distribution $W_{ji}^{\text{out}} \sim \sqrt{\frac{2.0}{n_{\text{rec}}}} \mathcal{N}(0,1)$, where n_{rec} is the number of neurons in the recurrent layer.

F.9. Training methods

The neuron fitting was performed using the L-BFGS-B algorithm [28], [29]. While for the task training, we utilized the online learning algorithm in BrainScale [41]. The integration time step Δt is 1 ms for the spiking neural network. The Adam optimizer [64] was used to calculate gradient-based optimization. The goal of the training was to minimize the cross-entropy between the output activity and the target output during the recall period.

G. Scalability analysis

The large-scale nature of the brain raises an important question: can our differentiable approach scale up to brain-scale spiking neural networks? Currently, we cannot provide a definitive answer, as it involves not only computational resource bottlenecks but also performance generalization to high-dimensional parameter spaces. However, from a computational resource perspective, we have evaluated how our approach can scale up the training of large-scale spiking networks with long time sequences (see Figure S6 for details).
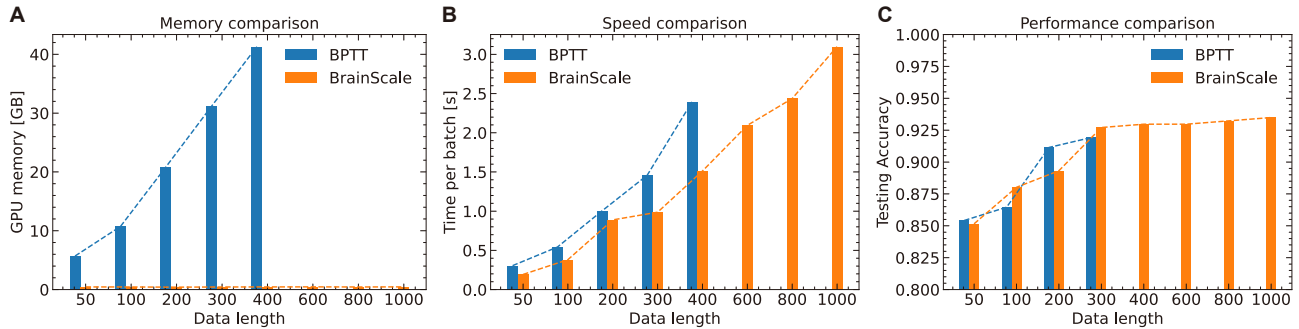

Figure S6. Comparative Analysis of Computational Memory, Speed, and Training Performance between BPTT and BrainScale [41] using the IBM DVS Gesture dataset [65]. (A) Memory Consumption Comparison per Batch: This subfigure illustrates the comparison of memory requirements between BPTT and BrainScale, showcasing the amount of GPU runtime memory utilized by each method for processing a single batch of data. (B) Computational Speed Comparison per Batch: This subfigure presents a comparative analysis of the computational speed achieved by BPTT and BrainScale for processing a single batch of data, highlighting the differences in their processing capabilities. (C) Maximum Testing Accuracy Comparison: This subfigure showcases the comparison of the maximum achieved testing accuracy between BPTT and BrainScale, emphasizing the performance differences of the two methods when evaluated on the IBM DVS Gesture dataset.

In particular, we conducted a memory and computational complexity analysis on a three-layer recurrent spiking neural network trained on the IBM DVS Gesture dataset [65]. With a batch size of 128 and 512 hidden neurons per layer, we compared the average memory usage and computation time per batch when training using backpropagation through time (BPTT) [66]–[68] and the BrainScale neuromorphic system [41] across various sequence lengths.

As shown in Figure S6A, the training memory required by BPTT increases linearly with the number of time steps, leading to an out-of-memory error for sequences longer than 600 steps. In contrast, BrainScale demonstrates remarkable memory efficiency by maintaining a constant memory footprint for the eligibility trace, regardless of the sequence length. Notably, BrainScale consumes less than 0.5 GB of GPU memory during the entire training process, reducing memory requirements by hundreds of times compared to BPTT.

The computational time for both BPTT and BrainScale scales linearly with the number of time steps (Figure S6B). However, BrainScale trains approximately twice as fast as BPTT, and this acceleration ratio increases for longer sequences due to its

event-driven and low complexity computation.

We also evaluated the training performance of BPTT and BrainScale (Figure S6C under the same hyperparameter settings for the training and network model. We calculated the maximum testing accuracy, and found that BrainScale demonstrated comparable performance to BPTT, even as the sequence length increased.

These results highlight the potential scalability of our differentiable approach, leveraging the computational efficiency and memory advantages of the online training system BrainScale. While further research is needed to generalize to larger networks and more complex tasks, our approach paves the way for training brain-scale spiking neural networks efficiently.

H. Limitations and potential challenges

The multi-scale differentiable brain modeling workflow described above is a comprehensive approach that aims to integrate various levels of information and constraints to build brain models that can reproduce cognitive behaviors observed in humans or animals. However, there are potential limitations and challenges to this approach:

1. *Data availability and quality*: The accuracy of brain models at each scale heavily depends on the quality and availability of the underlying data. At the microscale level, the accuracy of single neuron and synapse models relies on the quality and completeness of electrophysiological recording data. However, in practice, it is often challenging to obtain comprehensive electrophysiological data for all neurons. Similarly, at the mesoscopic level, the accuracy of connectome constraints depends on the quality and resolution of structural connectivity data obtained from techniques such as diffusion tensor imaging (DTI) or electron microscopy. These techniques face significant challenges in acquiring precise connectomic information in humans and animals. For example, while high-resolution electron microscopy can generate detailed 3D maps of neuronal connections from thin brain sections, the process is extremely labor-intensive, requires specialized equipment, and is currently limited to small tissue volumes due to the immense computational demands of reconstructing large-scale connectomes.

2. *Biological realism and simplifications*: While our approach aims to incorporate biological constraints, it necessarily involves simplifications and approximations of the underlying biological processes. In particular, our current model utilizes point-based simplified neuron models and considers only excitatory and inhibitory synaptic connections. This level of abstraction cannot fully capture the complexity and dynamics of real biological systems, especially when addressing intricate phenomena such as nonlinear dendritic effects, neuromodulation, and synaptic plasticity.

3. *Complexity and computational resources*: Building and training multi-scale brain models can be computationally intensive, particularly when dealing with large-scale networks and high-dimensional parameter spaces. To address this, we utilize the online learning framework BrainScale [41], which significantly reduces computational resource requirements for long-sequence learning tasks (see Appendix G). However, while online learning algorithms in BrainScale are memory-efficient, they are still approximations of full gradients and may encounter inefficiencies when training very high-dimensional parameters. Moreover, multi-scale brain models necessitate substantial computational resources, including powerful GPU hardware and efficient distributed computing algorithms. The complexity of these models demands not only high-performance computing infrastructure but also optimized software frameworks to manage and streamline the extensive computations involved.

4. *Integration across scales*: Seamlessly integrating information and constraints from different spatial and temporal scales is a non-trivial task. More biological details or constraints usually lead to more poor training performance. As more biological details are incorporated, the number of parameters in the model grows, making the optimization problem more complex and prone to issues such as vanishing/exploding gradients, local minima, and slow convergence.

5. *Interpretability and theoretical insights*: While the gradient-based optimization methods can produce models that fit the data, it may be challenging to derive theoretical insights or interpretable mechanisms from these models, especially when dealing with highly complex and non-linear brain dynamics models.

Despite these potential limitations and challenges, the multi-scale differentiable brain modeling workflow still represents a promising approach to integrate various levels of information and constraints to build more realistic and accurate brain models.
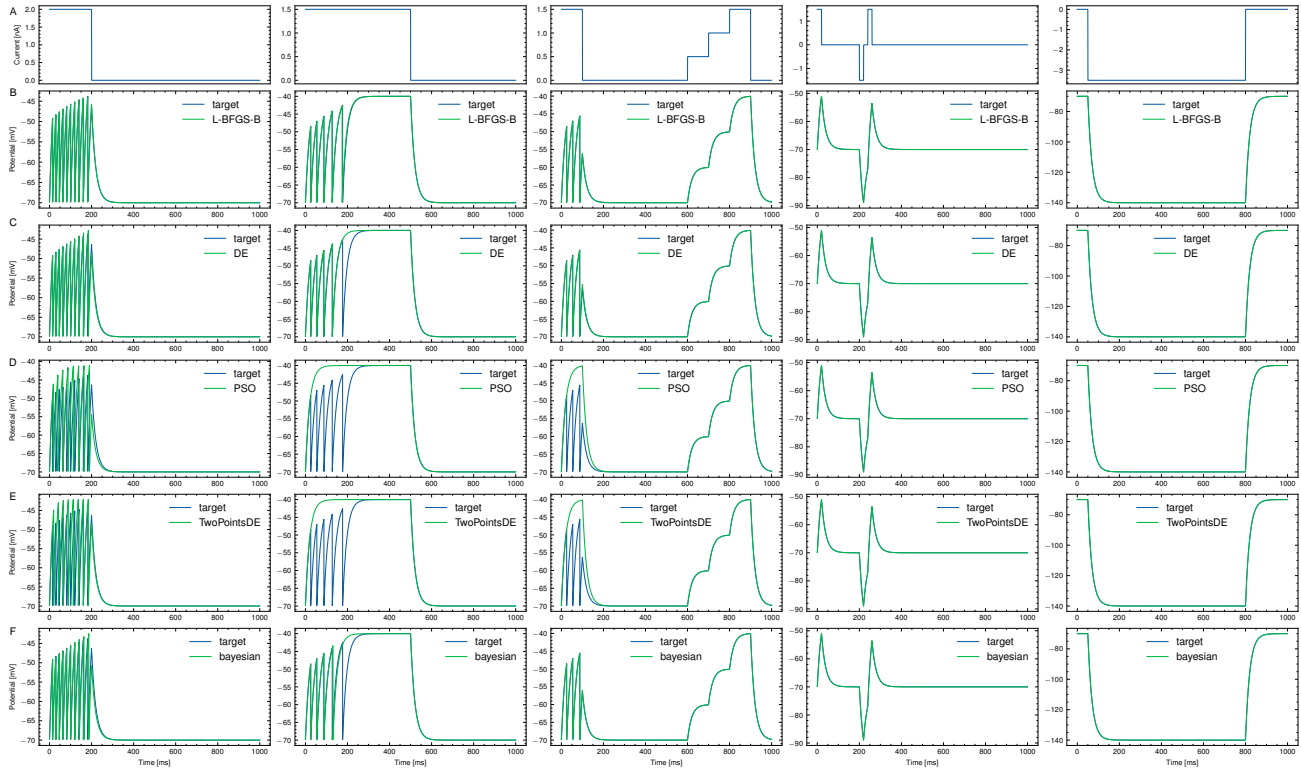
I. Supplementary data

Figure S7. Fitting the GIF neuron model to the membrane potential data. (A) Synaptic current synthesized in *vitor* for fitting the GIF neuron model with membrane potential data. (B) Fitting results of the GIF dynamics using the L-BFGS-B algorithm. (C) Fitting results of the GIF dynamics using the differential evolution (DE) algorithm provided in `Nevergrad`. (D) Fitting results of the GIF dynamics using the particle swarm optimization (PSO) algorithm provided in `Nevergrad`. (E) Fitting results of the GIF dynamics using the DE optimization with two points crossover (TwoPointsDE) algorithm provided in `Nevergrad`. (F) Fitting results of the GIF dynamics using the Bayesian optimization algorithm provided in `scikit-optimize`.
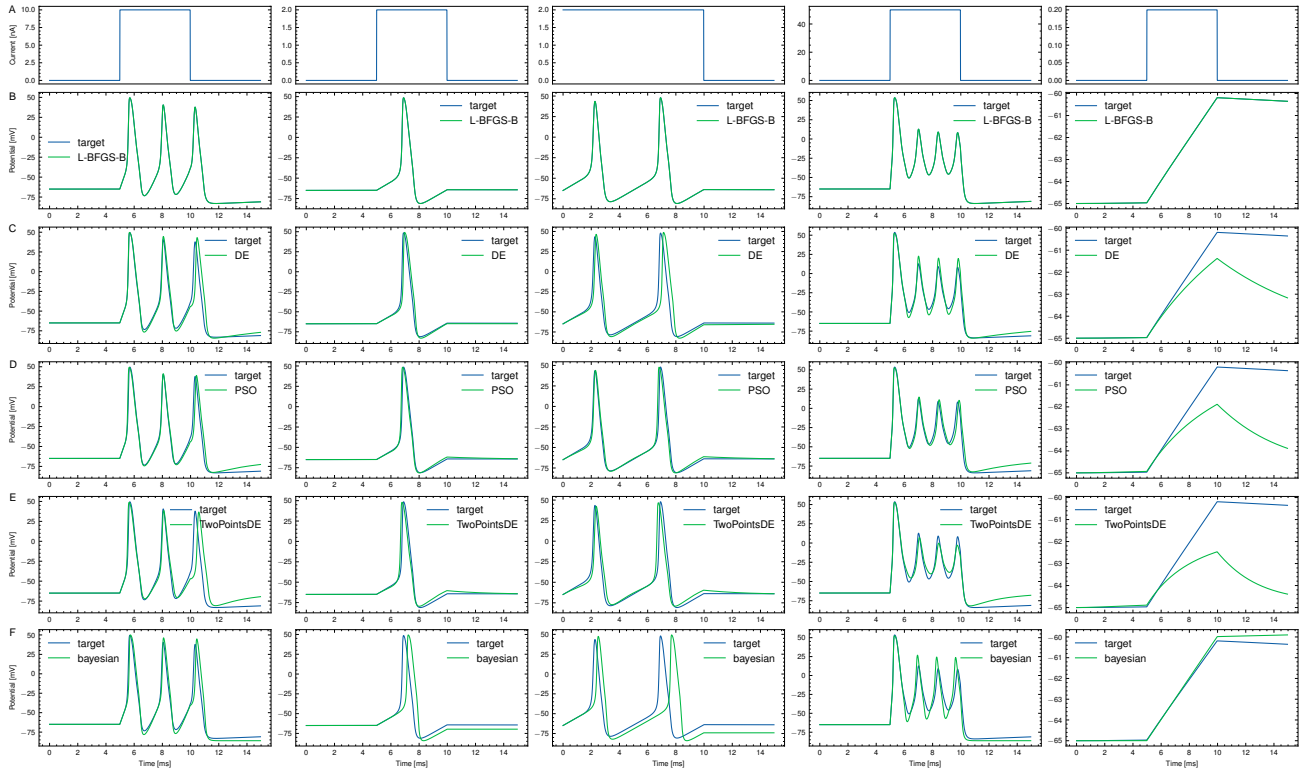
Figure S8. Fitting the HH neuron model to the membrane potential data. (A) Synaptic current synthesized in *vitor* for fitting the HH neuron model with membrane potential data. (B) Fitting results of the HH dynamics using the L-BFGS-B algorithm. (C) Fitting results of the HH dynamics using the differential evolution (DE) algorithm provided in `Nevergrad`. (D) Fitting results of the HH dynamics using the particle swarm optimization (PSO) algorithm provided in `Nevergrad`. (E) Fitting results of the HH dynamics using the DE optimization with two points crossover (TwoPointsDE) algorithm provided in `Nevergrad`. (F) Fitting results of the HH dynamics using the Bayesian optimization algorithm provided in `scikit-optimize`.

Table S2. The loss and speed comparison among five optimization methods, including L-BFGS-B, DE, PSO, TwoPointsDE, and Bayesian optimizations, when fitting the GIF neuron dynamics on the membrane potential data.

FITTING METHOD	FITTING LOSS	FITTING SPEED
L-BFGS-B (OURS)	6.799 ± 4.623	5.404 ± 0.294 s
DE (NEVERGRAD)	9.966 ± 2.281	1.172 ± 0.09 s
PSO (NEVERGRAD)	14.034 ± 2.857	1.151 ± 0.150 s
TWOPOINTSDE (NEVERGRAD)	9.702 ± 5.115	1.161 ± 0.172 s
BAYESIAN (SCIKIT-OPTIMIZE)	3.511 ± 1.741	62.330 ± 10.917 s

Table S3. The loss and speed comparison among five optimization methods, including L-BFGS-B, DE, PSO, TwoPointsDE, and Bayesian optimizations, when fitting the HH neuron dynamics on the membrane potential data.

FITTING METHOD	FITTING LOSS	FITTING SPEED
L-BFGS-B (OURS)	$2.3\text{E-}08 \pm 1.55\text{E-}08$	3.818 ± 0.725 s
DE (NEVERGRAD)	24.42 ± 22.55	0.619 ± 0.139 s
PSO (NEVERGRAD)	35.72 ± 24.06	0.663 ± 0.168 s
TWOPOINTSDE (NEVERGRAD)	32.31 ± 22.60	0.658 ± 0.175 s
BAYESIAN (SCIKIT-OPTIMIZE)	27.95 ± 25.76	55.26 ± 12.39 s