BrainScale: Enabling Scalable Online Learning in Spiking Neural Networks

Chaoming Wang[1,2], **Xingsi Dong**[1,2,3], **Jiedong Jiang**[4], **Zilong Ji**[5], **Xiao Liu**[1,2], **Si Wu**[1,2,3,*]
[1] School of Psychological and Cognitive Sciences, Peking University, Beijing, China
[2] PKU-IDG/McGovern Institute for Brain Research, Peking University, Beijing, China
[3] Center of Quantitative Biology, Peking-Tsinghua Center for Life Sciences, Academy for Advanced Interdisciplinary Studies, Peking University, Beijing, China
[4] Beijing International Center for Mathematical Research, Peking University, Beijing, China
[5] Institute of Cognitive Neuroscience, University College London, London, United Kingdom
[*] Corresponding author: `siwu@pku.edu.cn`

Abstract

Spiking neural networks (SNNs) have been recognized as a key approach for studying brain functions and realizing neuromorphic intelligence. However, a critical obstacle hindering their widespread application is the lack of a scalable online learning algorithm capable of supporting large-scale training of complex brain dynamics over behavioral timescales. Existing training algorithms are either limited to offline learning [46], impeding behavioral task training; constrained by high memory complexity [8, 6], hindering network scaling; or use oversimplified models [82] that fail to capture the intricate brain dynamics. Here, we introduce *BrainScale*, which achieves scalable online learning in SNNs across three levels. First, *BrainScale* accommodates the complex brain dynamics by supporting a wide spectrum of SNNs through a streamlined abstraction of synaptic interactions. Second, leveraging the inherent properties of SNNs, *BrainScale* simplifies the full gradient computation, realizing an online learning algorithm with linear memory complexity. Finally, *BrainScale* provides a program compiler that enables automatic derivation and execution of online learning computations for any user-defined model. Comprehensive evaluations demonstrate that compared to existing methods, *BrainScale* not only enables unprecedented complex and large-scale SNN training over behavioral timescales, but also generates more biologically realistic neural activity and synaptic connectivity in brain simulation models, as well as achieves comparable or even superior performance in neuromorphic tasks. These results highlight the potential of *BrainScale*-powered SNN online learning for widespread applications in brain simulation and neuromorphic computing.

1 Introduction

Spiking neural networks (SNNs), as the third generation of neural network models [36], have emerged as a pivotal approach for investigating brain cognitive functions [63, 7] and realizing neuromorphic artificial intelligence [60, 83]. By emulating the spiking mechanisms of biological neurons, these bio-inspired computational models offer a unique paradigm for understanding and replicating neural information processing in the brain [17, 38, 64]. In comparison to traditional artificial neural networks (ANNs), SNNs not only more accurately reflect the operational principles of biological neural systems [20] but also demonstrate significant advantages in energy efficiency and spatiotemporal information encoding [22]. Consequently, SNNs have garnered substantial attention in both brain science research [63, 7, 17] and neuromorphic computing efforts [60, 38, 64].

However, despite the conceptual appeal of SNNs, their widespread applications in brain simulation and neuromorphic computing still faces significant challenges. One of the most pressing issues is the lack of a scalable online learning algorithm. To address this deficiency, such an algorithm must satisfy several critical criteria:

1. *Scalable in Training Scope*: The algorithm should be adaptable to a diverse range of SNN models, considering the heterogeneity of neuronal types, the complexity of synaptic dynamics, and the intricate network topologies observed in biological neural systems.
2. *Scalable in Memory Efficiency*: The algorithm should demonstrate superior memory management, enabling online operation with a memory footprint that scales linearly with the number of neurons, thus facilitating the simulation of networks that approach the size of biological neural systems.
3. *Scalable in Implementation*: The algorithm should be supported by programming softwares capable of automatically deriving and solving online algorithms for user-defined SNN models, thereby accommodating flexible model designs and facilitating rapid prototyping and experimentation.

Developing such an algorithmic framework requires systematic breakthroughs in multiple domains (Figure 1), including not only mathematical innovations that reduce algorithmic complexity, but also model abstractions that balance biological plausibility and engineering practicality, and engineering advances in automated and general-purpose programming.

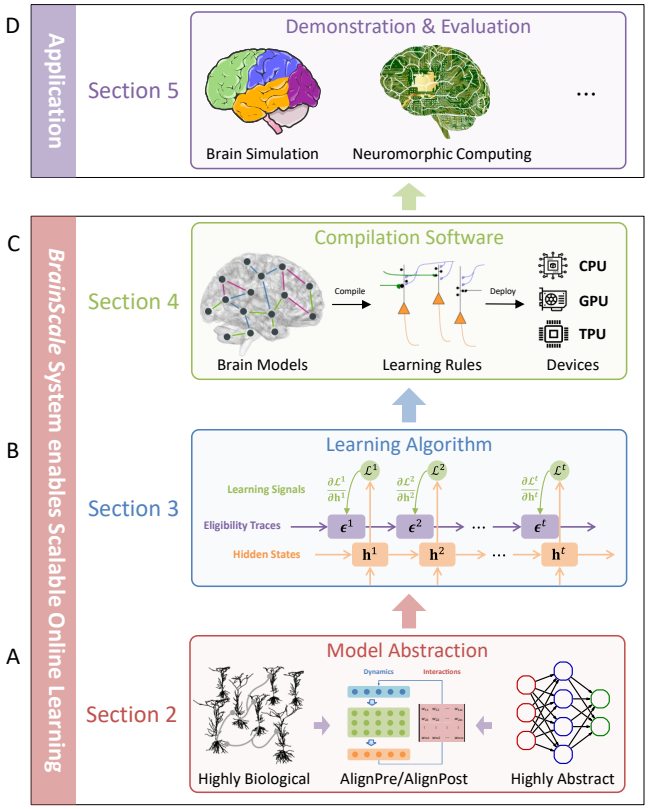

Figure 1: ***BrainScale*, a system for enabling scalable online learning in spiking neural networks.** (A) *BrainScale* introduces the `AlignPre` and `AlignPost` model abstractions, simplifying neuron interactions while balancing biological accuracy and engineering practicality. (B) By leveraging SNN properties, *BrainScale* reduces the complexity of full gradient computations, enabling memory-efficient online learning that scales with the number of neurons. (C) *BrainScale* features a compilation software that automatically derives the online learning computations for any user-defined SNN model and compiles them for deployment on CPUs, GPUs, and TPUs. (D) As a scalable SNN online learning system, *BrainScale* demonstrates broad applicability in both brain simulation and neuromorphic computing.

To this end, we introduce *BrainScale*, which provides systematic innovations to enable scalable online learning in SNNs (Figure 1). In terms of training scope, *BrainScale* supports training a wide range of SNN models that align with `AlignPre` and `AlignPost` model abstractions that simplify synaptic interactions (Figure 1A, Section 2). Regarding memory efficiency, *BrainScale* leverages the inherent characteristics of SNNs to reduce the complexity of full gradient computation, achieving online learning with memory usage that scales linearly with the number of neurons (Figure 1B, Section 3). In terms of implementation, *BrainScale* provides an automated compilation software that can automatically derive the online learning computation for any defined SNN model, regardless of its complexity (Figure 1C, Section 4).

We conduct extensive evaluations of *BrainScale* across various SNN models, neuromorphic datasets, and tasks (Figure 1D, Section 5). The results demonstrate that BrainScale achieves unprecedented memory efficiency: compared to traditional offline learning algorithms like backpropagation through time (BPTT) [46], *BrainScale* improves memory efficiency by a factor of $1/T$ (where T is the discretization time step); compared to existing online learning algorithms such as eprop [6] and OSTL [8], *BrainScale* shows a $1/N$ memory advantage (where N is the number of neurons). Furthermore, in brain simulation models, *BrainScale*'s trained neural activity and synaptic connectivity exhibit higher biological realism; while in neuromorphic computing models, *BrainScale* demonstrates comparable or even superior training performance in long-term dependency learning and neuromorphic tasks.

2 Spiking networks supported in *BrainScale*

Responding to requirement 1, *BrainScale* supports a large class of spiking networks, though not all types (Methods 7.1). These networks can be characterized by the following discretized dynamics:

$$\underbrace{\mathbf{h}^t = f(\mathbf{h}^{t-1}, t, \mathbf{I}^t)}_{\text{intrinsic dynamics}} \quad \text{and} \quad \underbrace{\mathbf{I}^t = \boldsymbol{\theta} \circledast \mathbf{x}^t}_{\text{neuronal interaction}} \tag{1}$$

where $\mathbf{h}^t \in \mathbb{R}^{Hd}$ represents the hidden states of H neurons, with each neuron quantified by d state variables; $\mathbf{I}^t \in \mathbb{R}^H$ denotes the synaptic currents; and t is the time. $\mathbf{x}^t \in \mathbb{R}^I$ represents all received synaptic inputs, such as presynaptic spikes; and $\boldsymbol{\theta} \in \mathbb{R}^P$ denotes the connection weights between pre- and postsynaptic neurons. One hallmark feature of these networks is the clear decomposition of the *intrinsic dynamics* and the *neuronal interaction* (see examples in Methods 7.2). The network intrinsic dynamics is described by d neuronal and synaptic variables that are all aligned to the dimension of neuron number H, and their evolution is governed by the element-wise activation function f. The neuronal interaction $\circledast$, parameterized by connection weights $\boldsymbol{\theta}$, describes how the presynaptic activity is transformed into postsynaptic currents, and can be implemented using sparse, dense, or convolutional kernels, depending on the network topology.

Eq. 1 is capable of modeling the exact dynamics of realistic brain networks under the assumption of homogeneous synaptic parameters. In real brains, when a single presynaptic neuron forms connections with k postsynaptic neurons, there are k distinct physical synapses requiring modeling for each of them (Figure 2A). However, if the synaptic parameters (e.g., time constants) are identical across these k synapses, they will exhibit identical dynamical trajectory driven by the same presynaptic neuron. This allows for the merging of synapses with the same parameters, necessitating only one shared synapse in modelling. From the postsynaptic perspective, synaptic dynamics triggered by k presynaptic neurons will be converged into a single postsynaptic neuron (Figure 2B). If these synapses exhibit exponential dynamics with the same time constants, they do not need to be modeled individually; instead, one shared exponential synapse can capture all synaptic inputs to the postsynaptic neuron. These simplifications are denoted as `AlignPre` and `AlignPost` projection, respectively. The former aligns the synaptic dynamics to the dimension of presynaptic neurons ($\mathbf{g}^{\text{pre}}$ in Figure 2C), and the later aligns to the postsynaptic dimension ($\mathbf{g}^{\text{post}}$ in Figure 2D). `AlignPre` and `AlignPost` hold distinct modelling advantages (Methods 7.1-7.2). `AlignPost` can only be applied to exponential-family synapses, but $\circledast$ can benefit from event-driven computation since $\mathbf{x}^t$ denotes the binary events. `AlignPre` can be applied to any synaptic dynamics but $\circledast$ requires conventional floating-value based operations.

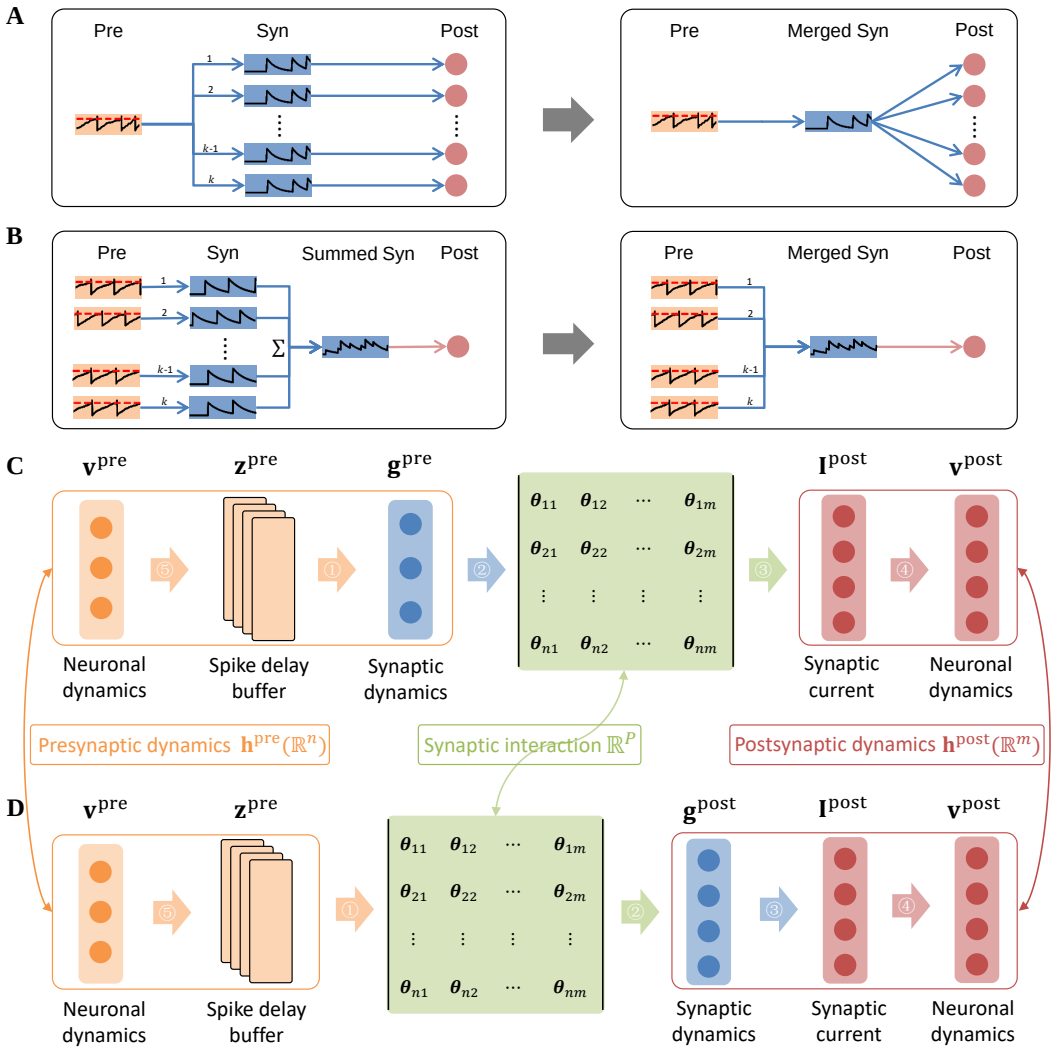

Figure 2: `AlignPre` **and** `AlignPost` **SNN abstractions.** (A) `AlignPre` modeling of a single presynaptic neuron, applicable to any synapse models. (B) `AlignPost` modeling of a single postsynaptic model, specifically applicable to exponential-family synapse models. (C) `AlignPre` modeling of a synaptic projection between two neuron populations. In `AlignPre`, the computation follows: Delayed presynaptic spikes $\mathbf{z}^{\text{pre}}$ drive updates to n synaptic dynamical variables $\mathbf{g}^{\text{pre}}$ ①; These dynamics determine postsynaptic conductances $\mathbf{g}^{\text{post}}$ according to the synaptic interaction ②; Then the postsynaptic neurons generate currents $\mathbf{I}^{\text{post}}$ by the conductance ③, which further drive the evolution of their neuronal dynamics $\mathbf{h}^{\text{post}}$ ④; Finally, postsynaptic neurons generate spikes $\mathbf{z}^{\text{post}}$ and update the spike delay buffer ⑤. (D) `AlignPost` modeling of a synaptic projection between two neuron populations. In `AlignPost`, the computation follows: Delayed presynaptic spikes are collected to determine conductance changes $\Delta\mathbf{g}^{\text{post}}$ for each postsynaptic neuron ①, which update the postsynaptic conductance variables $\mathbf{g}^{\text{post}}$ ②; These conductances then generate postsynaptic currents $\mathbf{I}^{\text{post}}$ ③ that update the postsynaptic neuron dynamics ④, ultimately resulting in postsynaptic spikes ⑤. (C-D) n and m represent the number of presynaptic and postsynaptic neurons, respectively; $\mathbf{v}$ represents neuronal dynamics; $\mathbf{g}$ denotes synaptic dynamics; $\mathbf{z}$ is the spike; and $\mathbf{I}^{\text{post}}$ denotes postsynaptic currents. The yellow, red, and green frames indicate presynaptic dynamics, postsynaptic dynamics, and synaptic interaction, respectively. For more details and concrete SNN networks, please refer to Methods 7.1 and 7.2.

3 Online learning algorithms in *BrainScale*

To solve the temporal credit assignment problem, the network must maintain eligibility traces $\boldsymbol{\epsilon}^t$ that track the potential synaptic strengthening or weakening based on history inputs and weight changes, until subsequent reward or punishment signals are received [19]. A complete tracking follows the update rule of real-time recurrent learning (RTRL) [81]:

$$\boldsymbol{\epsilon}^t = \frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} \boldsymbol{\epsilon}^{t-1} + \frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t}, \tag{2}$$

where the hidden Jacobian $\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}}$ governs eligibility propagation through time and the hidden-to-weight Jacobian $\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t}$ captures the influence of weight changes on the hidden state. Eq. 2 incurs prohibitive memory and computational overhead (Figure 3B, Methods 7.7.1). To achieve memory-efficient online learning (requirement 2), we exploit intrinsic SNN properties to reduce this complexity.

Using `AlignPre` and `AlignPost`, the hidden Jacobian of a spiking network is composed of two components: $\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \mathbf{D}^t + \mathbf{J}^t$, where $\mathbf{D}^t \in \mathbb{R}^{H \times H}$ is a diagonal matrix derived from the element-wise intrinsic dynamics, and $\mathbf{J}^t$ is a square matrix derived from the neuronal interaction (Eq. 1). Here, we set $d = 1$ to simplify the illustration of algorithmic deviations and the results can be generalized to $d > 1$ (Methods 7.7.4). SNNs are characterized by sparse activation patterns, with biological neurons typically firing 0.2-5 spikes per second [4] and only 1-2% of neurons in a cortical network active at any given moment [31]. The fact that $\mathbf{J}^t$ is highly sparse in both space and time suggests that $\mathbf{J}^t$ has limited influence and could potentially be ignored in the hidden Jacobian. We performed empirical evaluations across various SNN architectures (Methods 7.2) and found that a diagonal approximation ($\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} \approx \mathbf{D}^t$) achieves over 99% cosine similarity with the full hidden Jacobian (Figure 3A). This finding demonstrates that intrinsic neuronal dynamics dominate the hidden Jacobian in SNNs, and it aligns with earlier practices [6, 43] which show that neglecting gradient flow through $\mathbf{J}^t$ doesn't significantly impair learning performance in specific neuronal dynamics.

`AlignPre` and `AlignPost` also reflect physically constrained connections in biological networks, wherein neuronal interactions are followed by element-wise activations within individual neurons (as shown in Eq. 1 and Figure 2C-D). Such property leads to an important mathematical implication: the hidden-to-weight Jacobian $\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t}$ can be decomposed into a Kronecker product of a diagonal matrix and a vector (refer to Methods 7.4 for a detailed proof): $\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t} = \mathbf{D}_f^t \otimes \mathbf{x}^t$, where $\mathbf{D}_{f,ii}^t = \frac{\partial \mathbf{h}_i^t}{\partial \mathbf{I}_i^t}$ represents the diagonal entries of $\mathbf{D}_f^t$ and $\otimes$ denotes the Kronecker product.

Leveraging the above two properties, eligibility traces $\boldsymbol{\epsilon}$ in Eq. 2 can be reduced to the following rule:

$$\boldsymbol{\epsilon}^t \approx \mathbf{D}^t \boldsymbol{\epsilon}^{t-1} + \mathrm{diag}(\mathbf{D}_f^t) \otimes \mathbf{x}^t, \tag{3}$$

where $\mathrm{diag}(*)$ extracts the diagonal elements of the given matrix. We refer to Eq. 3 as Diagonal approximated RTRL (D-RTRL) for SNNs, and it shows $\mathcal{O}(BN^2)$ memory complexity (Figure 3C, Methods 7.7.2), where B is batch size. Although D-RTRL reduces complexity, its memory requirements still increase quadratically with the neuron number. Expanding the above iterative formula, we got $\boldsymbol{\epsilon}^t = \sum_{k=1}^t \frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^k} \approx \sum_{k=1}^t \mathbf{b}_k^t \otimes \mathbf{x}^k$, where $\mathbf{b}_k^t = \prod_{i=k+1}^t \mathrm{diag}(\mathbf{D}^i) \circ \mathrm{diag}(\mathbf{D}_f^k)$. This means that eligibility traces in D-RTRL are the summation of t rank-one matrices, where each matrix $\mathbf{b}_k^t \otimes \mathbf{x}^k$ captures the influence of $\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^k}$ as it propagates from time k to t (Figure S10). However, this form introduces significant memory overhead, as storing the sum requires maintaining t pairs of vectors.

Fortunately, we found that SNNs possess another unique characteristic that can overcome this overhead: the outputs of each neuron, corresponding to inputs $\mathbf{x}^t$, maintain a consistent sign across all time steps. In `AlignPost`, $\mathbf{x}^t$ denotes the binary spike vector, while in `AlignPre`, it represents the synaptic conductance (Methods 7.1). This property allows the transformation of the sum of multiple rank-one matrices into a single rank-one matrix between summed vectors. As a result, eligibility traces can be approximated as: $\boldsymbol{\epsilon}^t \approx \frac{1}{t} \sum_{k=1}^t \mathbf{b}_k^t \otimes \sum_{k=1}^t \mathbf{x}^k$. A theoretical proof, along with empirical experiments using various SNN dynamics on real-world neuromorphic datasets to validate this property, is provided in Methods 7.5. In practice, we compute the running average of $\mathbf{b}^t$

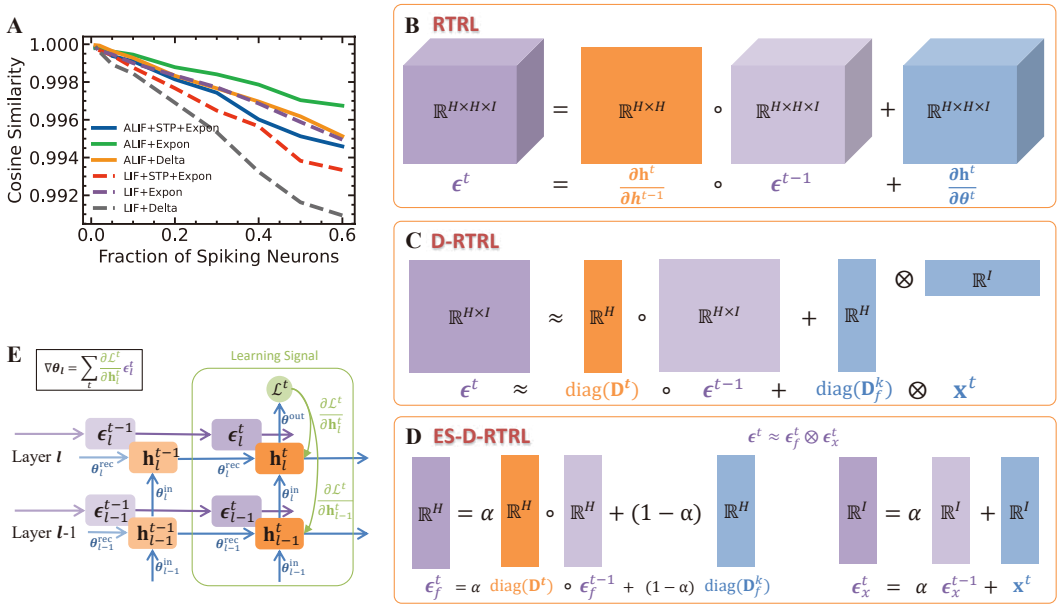

Figure 3: **Illustrations of online learning algorithms.** (A) Cosine similarity between the diagonal Jacobian $\mathbf{D}^t$ induced by intrinsic dynamics and the full hidden Jacobian $\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}}$ over different network dynamics and the fraction of spiking neurons in a single time-step. (B-D) Graphical illustration of the iterative eligibility traces RTRL [81] (B), D-RTRL (C), and ES-D-RTRL (D), respectively. (E) The online learning workflow of D-RTRL and ES-D-RTRL algorithms on multiple layers. Each layer computes its individual eligibility traces. When the learning signal, denoted by the green frame, from the output layer arrives at each layer at time t, the weight gradients $\nabla\boldsymbol{\theta}_l$ of the layer l are computed according to the learning rule Eq. 7.

and the running summation of $\mathbf{x}$, with the following eligibility update rule:

$$\boldsymbol{\epsilon}^t \approx \boldsymbol{\epsilon}_{\mathbf{f}}^t \otimes \boldsymbol{\epsilon}_{\mathbf{x}}^t, \tag{4}$$

$$\boldsymbol{\epsilon}_{\mathbf{x}}^t = \alpha \boldsymbol{\epsilon}_{\mathbf{x}}^{t-1} + \mathbf{x}^t, \tag{5}$$

$$\boldsymbol{\epsilon}_{\mathbf{f}}^t = \alpha\, \mathrm{diag}(\mathbf{D}^t) \circ \boldsymbol{\epsilon}_{\mathbf{f}}^{t-1} + (1-\alpha)\mathrm{diag}(\mathbf{D}_f^t), \tag{6}$$

where $\circ$ denotes element-wise multiplication, and α $(0 < \alpha < 1)$ is the smoothing factor corresponding to the time constant $\tau = -\Delta t/\ln(\alpha)$. We refer to Eqs. 4-6 as Exponentially Smoothed Diagonal approximated RTRL algorithm (ES-D-RTRL), in which $\boldsymbol{\epsilon}_{\mathbf{x}} \in \mathbb{R}^I$ and $\boldsymbol{\epsilon}_{\mathbf{f}} \in \mathbb{R}^H$ are eligibility traces tracking pre- and postsynaptic activity, respectively. Eligibility traces in ES-D-RTRL are operated temporally, locally, and in an unsupervised manner, resembling the classical learning rule with the form of spike-timing-dependent plasticity (STDP) [66]. Notably, it has $\mathcal{O}(BN)$ memory and computational complexity (Figure 3D, Methods 7.7.3).

D-RTRL and ES-D-RTRL can be applied to any supervised learning tasks. Consider scenarios where optimization targets are provided at specific times $t' \in \mathcal{T}$, generating a loss $\mathcal{L}^{t'}$ that measures the mismatch between targets and network outputs. Our aim is to optimize the total loss $\mathcal{L} = \sum_{t' \in \mathcal{T}} \mathcal{L}^{t'}$ with respect to parameters $\boldsymbol{\theta}$. Using the approximated eligibility traces above, the parameter gradient is computed as:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \sum_{t' \in \mathcal{T}} \frac{\partial \mathcal{L}^{t'}}{\partial \mathbf{h}^{t'}} \circ \boldsymbol{\epsilon}^{t'}, \tag{7}$$

where $\frac{\partial \mathcal{L}^{t'}}{\partial \mathbf{h}^{t'}} \in \mathbb{R}^{H \times 1}$ is the learning signal generated at time t'. Depending on the desired balance between biological plausibility and computational efficiency, learning signals can be generated using standard backpropagation through deep layers [61], random feedback alignment [33], or local losses per layer [57] (Figure 3E).

4 Online learning compiler in *BrainScale*

The success of backpropagation algorithms in deep learning is largely driven by programming systems like PyTorch [52] and TensorFlow [1], which can automatically compute necessary gradients. In parallel, scalable SNN learning require software that can automatically handle online learning computations (requirement 3). To address this need, we developed *BrainScale*, a program compiler that can automatically compile our proposed online learning algorithms to complex SNN dynamics and architectures. *BrainScale* was designed with the philosophy of "introducing minimal new concepts while maintaining a consistent workflow with established deep learning practices". It was built upon JAX [10] and BrainPy [75], leveraging JAX's automatic differentiation (AD) and just-in-time (JIT) compilation capabilities, as well as BrainPy's ecosystem for brain dynamics programming. A typical online learning workflow in *BrainScale* consists of three main steps.

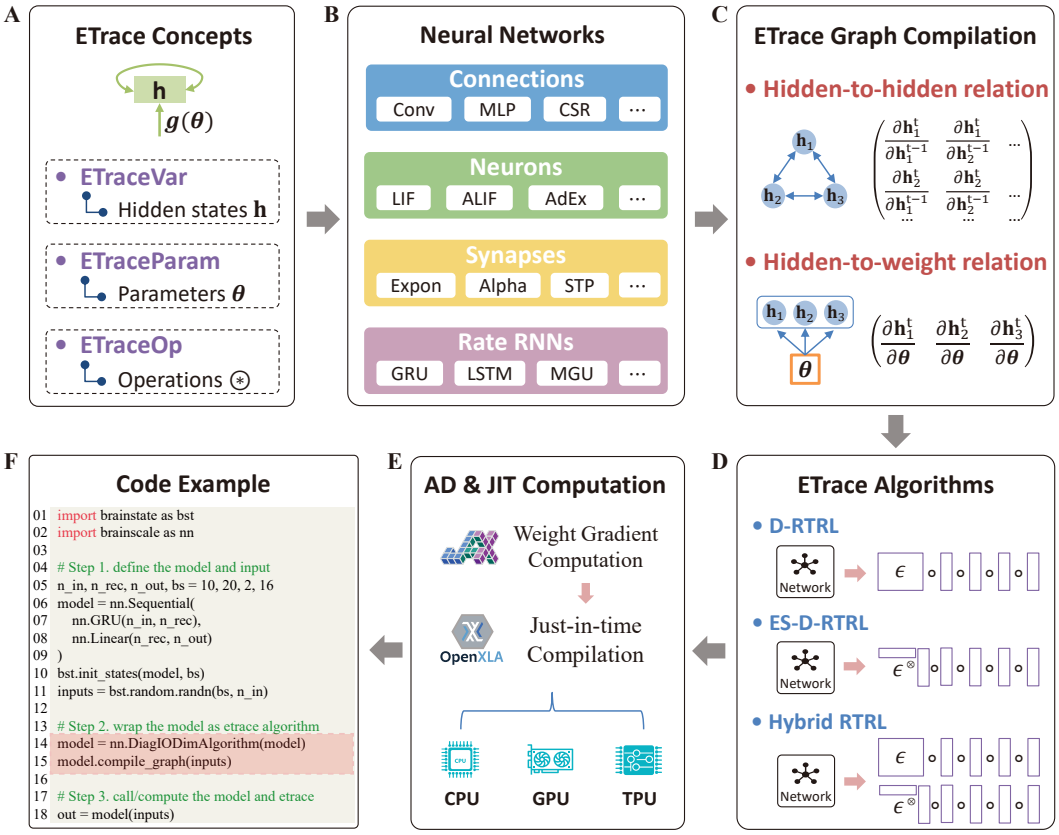

Figure 4: **The online learning program compiler.** (A-B) Step 1: defining hidden state updates. (A) New programming concepts for customizing the model for online learning. `ETraceVar`, `ETraceParam`, and `ETraceOp` are used to define hidden states, weight parameters, and input-to-hidden transformations, respectively. (B) Building blocks for defining high-level neural networks, including synaptic connections, neuronal dynamics, synaptic dynamics, and rate models. Other components please see the documentation of *BrainScale*. (C-D) Step 2: compiling eligibility traces ϵ. (C) Graph compilation for computing necessary components, including hidden-to-hidden and hidden-to-weight relationships, for eligibility trace. (D) Interfaces for defining eligibility trace algorithms. (E-F) Step 3: computing weight gradients $\nabla_\theta \mathcal{L}$. (E) Eligibility trace algorithms in *BrainScale* support reverse-mode AD in JAX for computing weight gradients and employ JIT compilation of XLA to deploy models on CPU, GPU, and TPU devices. (F) Code example to define an online learning model in *BrainScale*. The red box indicates optional code when online learning is not needed.

The first step involves defining how *hidden states* ($\mathbf{h}$) are updated based on the parameters ($\boldsymbol{\theta}$) and the input-to-hidden operations ($\circledast$) (Figure 4A-B). *BrainScale* offers two levels of interface for controlling model construction granularity. For customizing new components, *BrainScale* introduces three new concepts (Figure 4A): `ETraceVar` for defining hidden states $\mathbf{h}$, `ETraceParam` for trainable parameters

$\boldsymbol{\theta}$, and `ETraceOp` for input-to-hidden transformations $\circledast$. For composing components into large-scale, high-level networks, *BrainScale* provides rich building blocks (Figure 4B), including commonly used synapse connections, neuronal dynamics, synaptic dynamics, and deep learning components (e.g., normalizations, activations, and poolings).

The second step involves deriving *eligibility traces* ($\boldsymbol{\epsilon}$) based on the hidden state updates (Figure 4C-D). To automate this process without human intervention, *BrainScale* offers a robust graph compilation process. The core idea behind the compilation is to transform user-defined models into symbolically manipulable intermediate representations, facilitating easy analysis and extraction of hidden-to-hidden and hidden-to-weight relationships essential for computing eligibility traces (Figure 4C, Section 7.9). We first transform the Python-coded model into the Jaxpr intermediate representations in JAX [11], a fine-grained graph representing mathematical operations and data dependencies within the model. Then, *BrainScale* utilizes these Jaxpr representations to extracts high-level graphs of how each parameter $\boldsymbol{\theta}$ influences the hidden states $\mathbf{h}$ via the operation $\circledast$ (Figure 4A). The resulting `<hidden, weight, operation>` graph computes all data needed for eligibility traces, including the hidden-to-hidden Jacobian ($\mathrm{diag}(\mathbf{D}^t)$) and hidden-to-weight Jacobian ($\mathrm{diag}(\mathbf{D}_f^t)$ and $\mathbf{x}^t$). Moreover, *BrainScale* provides three out-of-the-box interfaces to define eligibility trace updates based on this high-level graph (Figure 4D): `DiagParamDimAlgorithm` uses the D-RTRL algorithm to update the traces, `DiagIODimAlgorithm` employs the ES-D-RTRL algorithm, and `DiagHybridDimAlgorithm` allows for selective updates using either the D-RTRL or ES-D-RTRL algorithms depending on user specifications or heuristic parameter rules.

The final step involves computing the *weight gradients* ($\nabla_{\boldsymbol{\theta}}\mathcal{L}$) based on the data from the previous two steps (Figure 4E-F). To facilitate seamless integration of online gradient computations with standard AD routines, we have made our online algorithms compatible with reverse-mode AD in JAX by customizing the vector-Jacobian product rule. This adaptation enables the online computation process to closely resemble that of traditional deep learning models, requiring no more than two lines of additional code (the red box in Figure 4F). To further accelerate online learning across modern hardware, we leverage JIT compilation via JAX and XLA to generate efficient machine codes for CPU, GPU, and TPU devices (Figure 4E).

By following these three steps, *BrainScale* ensures a streamlined and efficient process for scalable online learning in SNNs. The simplicity of this online learning process is highlighted in Methods C.

5 Evaluations for *BrainScale*

5.1 Evaluations on approximation, long-term dependency, memory, and speed

We first evaluate gradient approximation, long-term dependency learning, memory usage, and running speed of our online learning framework across diverse network dynamics and neuromorphic datasets.

Approximation accuracy of hidden Jacobian and weight gradient. The *BrainScale* platform incorporates different approximations for computing parameter gradients in its online learning algorithms. To assess the effectiveness of these algorithms in approximating the hidden Jacobian and weight gradients on real-world neuromorphic datasets, we conducted empirical experiments using IBM DVS Gesture [3], Spiking Heidelberg Digits (SHD) [15], and Neuromorphic-MNIST (N-MNIST) [48] datasets. We explored various SNN dynamics, ranging from simple leaky integrate-and-fire (LIF) to adaptive LIF (ALIF) neurons, and synapses with diverse dynamics including Delta, exponential, short-term depression (STD), and short-term plasticity (STP) (Methods 7.2).

We first evaluated the cosine similarity between the single-step hidden Jacobian ($1/T\sum_i^T \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i}$) and the long-range hidden Jacobian ($\frac{\partial \mathbf{h}^T}{\partial \mathbf{h}^1}$), where T is the length of each dataset. Our analysis revealed consistently high approximation accuracy for both single-step and long-range hidden Jacobians (Figure 5A-B). Furthermore, we found that high cosine similarities typically favor networks exhibiting complex intrinsic dynamics (refer to ALIF-based networks in Figure 5B).

We then assessed weight gradient approximations by comparing the cosine similarity between gradients computed by our online learning algorithms and the offline algorithm BPTT. The gradients computed by D-RTRL exhibited a cosine similarity (Figure 5C) comparable to its hidden Jacobian approximation (Figure 5A). ES-D-RTRL, however, demonstrated a relatively lower gradient similarity

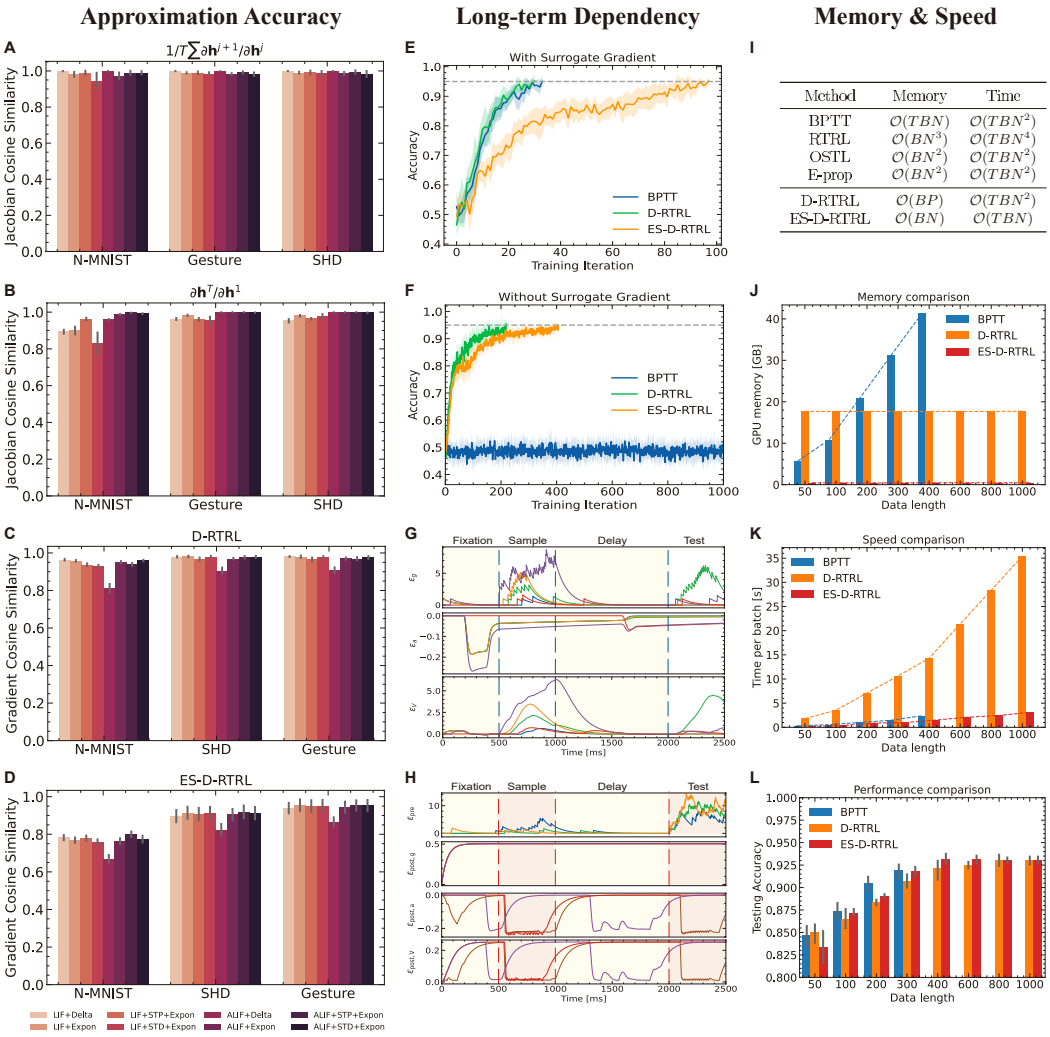

Figure 5: **Evaluation of learning algorithms on approximation accuracy, long-term dependency learning, memory usage, and computational speed**. (A-D) Panels A-D show results across three neuromorphic datasets [3, 15, 48] and eight network dynamics. For each dataset, we randomly selected 500 data points, computed the Jacobian cosine similarity for each data point, and reported the average cosine similarity across all samples. (A) Cosine similarity between approximated and exact single-step hidden Jacobians: $1/T\sum_i^T \partial\mathbf{h}^{i+1}/\partial\mathbf{h}^i$. (B) Cosine similarity between approximated and exact long-range hidden Jacobians: $\partial\mathbf{h}^T/\partial\mathbf{h}^1$. (C) Cosine similarity between weight gradients computed by D-RTRL and BPTT algorithms. (D) Cosine similarity between weight gradients computed by ES-D-RTRL and BPTT algorithms. (E-H) Evaluation of long-term dependency learning capability using the DMTS task. (E) Comparison of testing accuracy for different algorithms on networks utilizing surrogate gradients. (F) Comparison of testing accuracy for different algorithms on networks without surrogate gradients. (G) Visualization of the eligibility trace in the D-RTRL algorithm, showing data from five randomly sampled positions. (H) Visualization of the eligibility trace in the ES-D-RTRL algorithm, displaying data from three randomly sampled positions. (I-L) Comparison of memory efficiency and running speed between different learning algorithms. All evaluations were performed using the Gesture dataset [3]. (I) Theoretical analysis of memory and computational complexity. (J) Comparison of GPU memory consumption between our online learning algorithms and BPTT during one batch of training. (K) Comparison of training speed for one batch between our online learning algorithms and BPTT during one batch training. (L) Comparison of final testing accuracy over different data lengths.

compared to D-RTRL (Figure 5D), due to the additional approximations involved in computing eligibility traces (Methods 7.7.3). Moreover, both algorithms exhibited lower gradient similarities on the N-MNIST dataset (Figure 5C-D), indicating some dataset dependence. Despite this, both D-RTRL and ES-D-RTRL gradients maintained high approximation, demonstrating their effectiveness in capturing essential gradients for training SNNs on neuromorphic datasets. Further investigation revealed several key insights about factors affecting the approximation performance (Appendix A). Network size showed no significant impact on approximation accuracy (Figure S12). However, approximation accuracy declined greatly when either feedforward (Figure S13) or recurrent (Figure S14) weight strength increased to levels that produced unrealistically high firing rates. Additionally, a deeper recurrent layers led to lower approximation accuracies (Figure S15), suggesting a preference for shallow network architectures in our algorithms.

Long-range dependency learning. Spiking networks hold promise for modeling long-term dependencies in datasets. Consequently, a critical requirement for SNN training is the ability to assign credit for these long-term dependencies. Our online learning algorithms trace gradient dependencies based solely on the intrinsic dynamics of each neuron and synapse. We aim to evaluate whether this partial information suffices for learning very long sequences.

We first trained recurrent spiking networks (Eq. 46) on the delayed match-to-sample (DMTS) task [14], a task requiring the network to maintain directional information from the sample phase over a long delay period (>1000 ms) before making a decision based on the test period stimulus (Methods 7.8.4). We initially applied surrogate spiking gradients [46] (Methods 7.8.1) on the network training. Experiments demonstrated that D-RTRL, ES-D-RTRL, and BPTT all effectively solve the task, with BPTT and D-RTRL showing faster convergence compared to ES-D-RTRL (Figure 5E). However, without surrogate gradients, BPTT failed to train successfully on this task, with its training accuracy fluctuating around the 50% random baseline (Figure 5F). In contrast, D-RTRL and ES-D-RTRL were capable of learning normally without compromising training accuracy. These findings highlight that utilizing intrinsic dynamics to train spiking networks can effectively capture long-term dependency information from spiking data. To further verify the long-term dependency learning capability of our online learning algorithms, we conducted experiments on gated recurrent units (GRU) [13] using the classical copying task [25]. This task requires the network to reproduce the first 10 tokens after processing N tokens at the final 10 time steps. We trained our models using D-RTRL and BPTT, excluding ES-D-RTRL as GRU does not satisfy its requirements. For $N = 100$, D-RTRL exhibited lower initial loss (Figure S16A), indicating rapid extraction of long-term dependency information. This trend persisted for longer sequences: for $N = 300$ (Figure S16B) and $N = 500$ (Figure S16C), BPTT failed to converge, while D-RTRL continued to learn. However, in later stages of training for $N = 100$, D-RTRL's loss plateaued and converged more slowly than BPTT (Figure S16A). This suggests that while D-RTRL excels at extracting long-term dependency information, BPTT achieves lower loss values once it learns the task.

The capability of our online learning algorithms to learn long-term dependencies raises important questions. The first is how our algorithms achieve long-term dependency learning. To investigate this, we analyzed the eligibility traces during the network's state propagation in the DMTS task. D-RTRL maintains three eligibility traces: $\boldsymbol{\epsilon}_{\mathbf{g}}$, $\boldsymbol{\epsilon}_{\mathbf{a}}$, and $\boldsymbol{\epsilon}_{\mathbf{v}}$, corresponding to the hidden states $\mathbf{g}$, $\mathbf{a}$, and $\mathbf{v}$ (Eq. 46). Visualization at five randomly selected positions (Figure 5G) revealed distinct behaviors: $\boldsymbol{\epsilon}_{\mathbf{g}}$ tracks spiking inputs, increasing upon arrival of spiking stimuli and gradually decaying during the delay period; $\epsilon_{\mathbf{a}}$ tracks the influence Jacobian with slow dynamics, facilitating information transfer across the delay; $\boldsymbol{\epsilon}_{\mathbf{v}}$ integrates information tracked from $\boldsymbol{\epsilon}_{\mathbf{g}}$ and $\boldsymbol{\epsilon}_{\mathbf{a}}$, exhibiting sensitivity to input changes as $\boldsymbol{\epsilon}_{\mathbf{g}}$ while gradually decaying as $\boldsymbol{\epsilon}_{\mathbf{a}}$. In contrast, ES-D-RTRL exhibits markedly different learning behavior (Figure 5H). The presynaptic eligibility trace $\boldsymbol{\epsilon}_{\mathrm{pre}}$ behaves similarly to $\boldsymbol{\epsilon}_{\mathbf{g}}$ in D-RTRL but is driven solely by presynaptic spikes. This distinction is also evident in $\boldsymbol{\epsilon}_{\mathrm{post},\mathbf{g}}$, which tracks postsynaptic current information and maintains a constant value throughout. But similar to D-RTRL, $\boldsymbol{\epsilon}_{\mathrm{post},\mathbf{a}}$ and $\boldsymbol{\epsilon}_{\mathrm{post},\mathbf{v}}$ exhibited slow dynamics, delivering long-term postsynaptic Jacobian information over a long time, until the arrival of learning signals during the test period. Another critical question concerns the differences in synaptic connectivity learned by our online learning algorithms compared to classical BPTT. Analysis of synaptic weights post training revealed that both D-RTRL and ES-D-RTRL maintained similar weight distributions with BPTT, yet they displayed distinct column structures, particularly in recurrent weights (Figure S17). This pattern, where incoming synaptic weights for a subset of postsynaptic neurons tend to be consistently stronger or weaker together, largely stems from our online algorithms' learning mechanism. Interestingly, similar

column structures have been observed in the synaptic connectome of layer 4 in the somatosensory cortex (Figure S18A, adapted from Fig. 3E in [40]), suggesting that our online learning algorithms may capture biologically relevant connectivity patterns.

Memory efficiency and computational speed. Finally, we evaluated the memory and computational complexity of D-RTRL and ES-D-RTRL in comparison to BPTT and other online learning algorithms. Figure 5I summarize the theoretical memory and time complexity. In general, D-RTRL has a memory complexity proportional to the parameter size P, achieving $\mathcal{O}(BP)$ complexity. ES-D-RTRL achieves a memory complexity of $\mathcal{O}(BN)$, scaling linearly with the number of neurons in the network because it only needs to store two traces of presynaptic and postsynaptic activities. Both BPTT and D-RTRL show a theoretical computational complexity of $\mathcal{O}(TBN^2)$, but differ in practice: BPTT involves a computation-bound matrix-matrix multiplication, while D-RTRL employs a memory-bound batched matrix-vector element-wise multiplication (Figure 3C and Eq. 3). ES-D-RTRL has a especially lower computational complexity of $\mathcal{O}(TBN)$, involving element-wise operations among presynaptic and postsynaptic neuronal activities.

We further conducted empirical experiments using the IBM DVS Gesture dataset [3] across various time steps. The network architecture consisted of three recurrent layers, each with 512 LIF neurons interconnected through dense connections with Delta synaptic dynamics (Eq. 43). The input layer received data of size 32,768, corresponding to an image shape of (128, 128, 2). Using a training batch size of 128, we calculated the average memory consumption and computation time per batch on a GPU. Our findings, illustrated in Figure 5J, showed that BPTT's GPU memory usage increased linearly with the number of time steps, reaching its limit at 600 time steps due to an out-of-memory error. In contrast, D-RTRL and ES-D-RTRL demonstrated significant memory efficiency, requiring only constant memory to maintain the eligibility trace regardless of data processing time length. ES-D-RTRL consumed less than 0.5 GB of GPU runtime memory throughout training, while D-RTRL required approximately 17.5 GB. Overall, ES-D-RTRL achieved hundreds-fold better memory efficiency than BPTT and 35-fold better than D-RTRL. Computational time evaluations (Figure 5K) revealed that all three algorithms' time consumption scaled linearly with the number of time steps. However, ES-D-RTRL ran significantly faster, being roughly twice as fast as BPTT and achieving ten times the acceleration compared to D-RTRL. This acceleration ratio became more pronounced as sequence length increased. We also recorded training performance during our evaluations. Figure 5L presents the mean testing accuracy of ten evaluations, with a 95% confidence interval. All three algorithms performed better as data sequence length increased. BPTT outperformed D-RTRL and ES-D-RTRL when the data length was 200, while at other lengths, D-RTRL and ES-D-RTRL showed comparable performance. Notably, in this classification task, ES-D-RTRL exhibited slightly better training performance than D-RTRL as data length increased.

5.2 Evaluations on SNNs for brain simulation

Our online learning algorithms are theoretically capable of supporting a large class of spiking networks, including those that mimic realistic brain networks with diverse neuronal types and complex synaptic dynamics (Section 2). To illustrate this, we developed a biologically-informed spiking network incorporating key features of realistic neural circuits: distinct excitatory and inhibitory (E/I) neuron populations, conductance-based synapses [74], and sparse connectivity patterns based on known connectome data [69]. Specifically, the network consists of 800 ALIF neurons, with a 4:1 ratio of excitatory to inhibitory neurons. Neurons were connected with a random probability ratio of 20%, according to the connection probabilities found in neocortex [37]. Each connection followed a conductance-based exponential synapse dynamics (Eq. 52). We trained this network on an evidence accumulation task [39], in which animals navigate a linear track in a virtual environment and are presented with seven visual cues (Figure 6A, Methods 7.8.5). The task required animals to count visual cues on the left or right and, upon reaching a T-junction, decide whether to turn left or right. Correct decisions, based on the majority of visual cues, were rewarded.

We conducted training both with and without surrogate gradient functions (Methods 7.8.1). The experiments corroborated the results observed in the DMTS task: BPTT, D-RTRL, and ES-D-RTRL all demonstrated high performance when utilizing surrogate gradients (Figure 6B). Notably, we found that D-RTRL outperformed BPTT in this task, exhibiting a steeper learning curve. ES-D-RTRL, while effective, required a longer training period to achieve comparable accuracy levels. The removal of surrogate gradients revealed critical differences in algorithm robustness (Figure 6C). BPTT failed

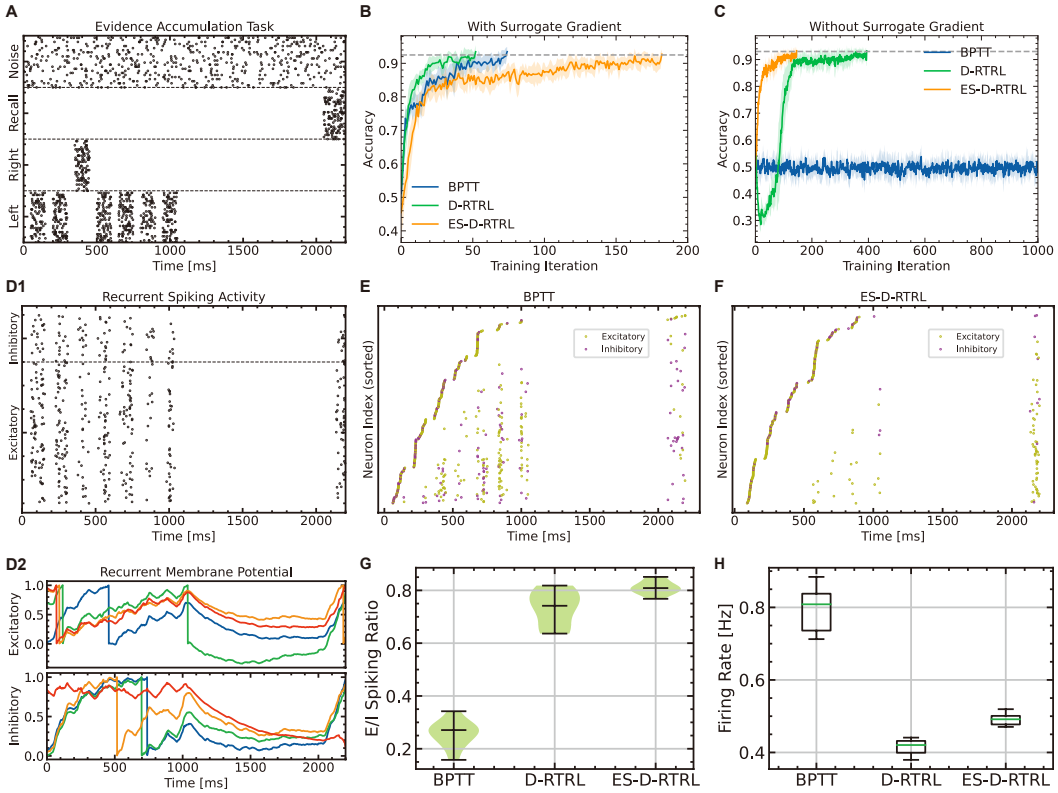

Figure 6: **Training and analysis of biologically-informed excitatory-and-inhibitory network for cognitive task performance**. (A) An example of spiking inputs demonstrating the evidence accumulation task. The x-axis denotes time, and the y-axis represents the neuron index. (B) Comparison of testing accuracy for BPTT, D-RTRL, and ES-D-RTRL algorithms when training networks with surrogate gradients. (C) Comparison of testing accuracy for BPTT, D-RTRL, and ES-D-RTRL algorithms when training networks without surrogate gradients. (D) Network activity after training with the ES-D-RTRL algorithm, in response to the spiking input train illustrated in (A). (D1) Spiking activities of recurrent neurons. (D2) Membrane potentials of five excitatory and inhibitory neurons in the recurrent layer. (E) Sorted raster plot of recurrent spiking in the network after training with BPTT. (F) Sorted raster plot of recurrent spiking in the network after training with ES-D-RTRL. (G) Excitatory-to-inhibitory spiking ratio during the recall period. (H) Firing rate comparison of the three algorithms over the entire period.

to learn the task without surrogate gradients. In contrast, both D-RTRL and ES-D-RTRL maintained their ability to learn, albeit with altered performance profiles. D-RTRL experienced a performance decline in the absence of surrogate gradients. However, ES-D-RTRL not only retained its performance without surrogate gradients but even showed slight improvement, suggesting that it may be inherently more robust to the discrete nature of spike-based computation in this kind of networks.

Analysis of trained networks showed consistent behavior in both excitatory and inhibitory neuron groups. Figure 6D1 and Figure S20A illustrate that neurons in both groups exhibited equally strong responses to stimuli throughout the task, indicating effective information processing within the network's excitatory-inhibitory circuit. Further examination of membrane potential dynamics showed that, unlike simplified models commonly used in deep learning, our conductance-based synapses produced more realistic behaviors. As shown in Figures 6D2 and S20B, membrane potentials were naturally constrained within excitatory and inhibitory reversal potentials, a fundamental characteristic of conductance-based models that removes the need for artificial techniques often used in other neural network applications.

We further analyzed the similarities and differences between networks trained by BPTT and online methods. We found significant differences in neuronal response patterns (Figure 6E-H). During

evidence accumulation phases (0 - 1100 ms), neurons trained by online algorithms exhibited high selectivity to specific visual cues. Sequentially presented visual cues induced orderly transitions between transient and clustered neuron population activities. Neurons in each population predominantly utilized a time-to-first-spike coding strategy for sensory processing. Notably, within each selective population, neuronal responses to external stimuli displayed clear rank ordering, with neurons firing sequentially in response to stimuli (Figure 6F and Figure S20). This temporal coding scheme suggests an efficient information encoding strategy that capitalizes on the precise timing of spikes [73]. Interestingly, these neural dynamics closely align with activities observed in the posterior parietal cortex of mice performing similar evidence accumulation tasks (Figure S18B, adapted from Fig. 2a in [39]). In contrast, neurons in BPTT-trained networks demonstrated reduced selectivity to sensory cues. Neurons that initially spiked in response to one sensory stimulus had a high probability of spiking when presented with other stimuli (Figure 6E). This pattern indicates a more distributed representation of sensory information in BPTT-trained networks. Interestingly, during the recall period, neuronal selectivity did not differ significantly between networks trained by either BPTT or online algorithms. We observed that most neurons responding to the recall cue. However, a notable distinction emerged in the type of neurons engaged: in networks trained by online algorithms, the majority of responsive neurons during recall periods were excitatory, whereas in BPTT-trained networks, they were predominantly inhibitory (Figure 6G). This difference in excitatory-inhibitory balance during recall may reflect distinct strategies for information retrieval and decision-making. We further analyzed the firing rate of network trained by different algorithms. Networks trained by BPTT generated more spikes overall to accomplish the task compared to those trained by online algorithms (Figure 6H).

5.3 Evaluations on SNNs for neuromorphic computing

Spiking networks have recently attracted substantial interest in the field of BIC due to their superior capability to replicate the structure, function, and dynamics of the human brain over traditional artificial neural networks [60]. To evaluate the effectiveness of our online learning algorithms for BIC applications, we conducted extensive tests using SNN models on several standard sequential benchmarks. These benchmarks, including N-MNIST [48], SHD [15], and IBM DVS Gesture [3], are specifically designed to assess the performance of spiking networks in processing time-varying spiking data.

Dataset	Training Method	Network Architecture	Testing Accuracy
N-MNIST [48]	BPTT [53]	Recurrent SNN	97.50%
	SLAYER [65]	Recurrent SNN	99.20%
	ETLP [55]	Recurrent SNN	94.30%
	OSTL [8]	Recurrent SNN	96.80%
	E-prop [6]	Recurrent SNN	97.90%
	BPTT (ours)	Recurrent SNN	97.90%
	D-RTRL (ours)	Recurrent SNN	98.59%
	ES-D-RTRL (ours)	Recurrent SNN	98.64%
SHD [15]	ETLP [55]	Recurrent SNN	74.59%
	E-prop [6]	Recurrent SNN	80.79%
	OSTTP [49]	Recurrent SNN	77.33%
	BPTT [59]	Convolutional SNN	83.50%
	BPTT [53]	Recurrent SNN	82.70%
	BPTT (ours)	Recurrent SNN	80.91%
	D-RTRL (ours)	Recurrent SNN	83.92%
	ES-D-RTRL (ours)	Recurrent SNN	84.62%
Gesture [3]	BPTT [59]	Convolutional SNN	86.70%
	BPTT [53]	Recurrent SNN	82.10%
	SLAYER [65]	Recurrent SNN	93.64%
	BPTT (ours)	Recurrent SNN	92.97%
	D-RTRL (ours)	Recurrent SNN	93.75%
	ES-D-RTRL (ours)	Recurrent SNN	94.27%

Table 1: Accuracy comparison between our online learning algorithms and prior methods.

Our experimental setup involved a three-layer spiking network, with each layer consisting of 200 LIF neurons interconnected via Delta synapses (Eq. 43). To ensure a fair comparison, we also trained this network using BPTT under identical hyper-parameter settings.

Table 1 presents a summary of our model's training results, juxtaposed with existing state-of-the-art online and offline SNNs. Our findings revealed that the models trained with our online learning algorithms consistently and significantly outperform SNNs trained using offline BPTT approaches and other online methods, including OSTL [8], E-prop [6], ETLP [55], SLAYER [65], and OSTTP [49].

An intriguing observation emerged from our experiments: despite the gradient computed by ES-D-RTRL demonstrating a lower cosine similarity to that calculated by BPTT compared to D-RTRL (Figure 5C-D), it consistently achieved superior performance on these neuromorphic datasets. This unexpected result suggests that the relationship between gradient similarity and task performance may be more complex than initially assumed, particularly in the context of spiking networks and neuromorphic data.

6 Discussion

In conclusion, we developed *BrainScale* to enable scalable online learning in spiking networks. *BrainScale* provides systematic innovations to meet the following essential criteria for scalability (Figure 1):

- *Training Scope*: By making minimal assumptions about synaptic interactions (the source of complexity in brain simulation), *BrainScale* significantly reduces model complexity while still covering the vast majority of neuron and synapse dynamics (Figure 1A, Section 2).
- *Memory efficiency*: We found that spiking networks have unique properties that can reduce online learning computations to $\mathcal{O}(N)$ memory scaling, a complexity which is crucial for training brain-scale SNNs that approach the size of biological neural systems (Figure 1B, Section 3).
- *Implementation*: *BrainScale* applies compilation techniques to automatically analyze, derive, and compute online learning processes, facilitating wide adoptions of SNNs on various applications (Figure 1C, Section 4).

Our comprehensive evaluations demonstrate that *BrainScale* not only approximates gradients effectively but also significantly enhances memory efficiency and computational speed in comparison to existing methods. Its robust performance across tasks involving long-term dependencies and neuromorphic datasets further validates its potential for large-scale brain simulation and neuromorphic computing in real-world applications (Figure 1D, Section 5). What's more, *BrainScale* can be seamlessly integrated with our previously developed general-purpose brain simulator, BrainPy [76]. As a result, *BrainScale* is positioned to fully unleash the potential of SNN online learning for solving complex tasks in both brain simulation and neuromorphic computing domains.

A key feature of our online learning algorithms is their ability to learn long-term dependencies in data by leveraging only the intrinsic dynamics of neuronal and synaptic models, without relying on neuronal interaction. We demonstrated that this approach is not only sufficient for long-term dependency learning in spiking networks (Figure 5) but also outperforms BPTT in rate GRU models (Figure S16). This result underscores the crucial role of intrinsic dynamics in long-term dependency learning. It also aligns with three decades of research on gated RNNs [25, 13, 50, 21], which have consistently shown that preserving long-range dependencies in RNNs requires the incorporation of element-wise linear gated recurrence. This gated recurrence can be viewed as a special form of intrinsic dynamics in brain simulation models (Appendix B). This connection raises an important question: can we further enhance the long-range memorization capability of the RNNs by drawing insights from complex, realistic neuron simulations? The answer is likely yes. Current RNNs are primarily inspired by single-compartment point neurons, whereas realistic neurons possess multiple compartments and rich dynamics [35]. Numerous studies have shown that dendritic cells exhibit remarkable computational capabilities that significantly influence neuronal and circuit function in ways that point neurons cannot capture [54], including local nonlinear transformation, synaptic modulation, and neural signal multiplexing. Integrating these complex dynamics into RNNs could improve their ability to learn multi-timescale dynamics and enhance long-term memory.

Our ES-D-RTRL algorithm can be interpreted as a three-factor Hebbian learning rule [19], where weight updates are are guided by two types of neural signals: eligibility traces $\boldsymbol{\epsilon}^t$ (Eqs. 4-6) and learning signals $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}$ (Eq. 7). Eligibility traces are regulated by two factors: the intrinsic dynamics of postsynaptic neurons $\mathrm{diag}(\mathbf{D}_t)$ and the factor α. $\mathbf{D}_t$ indicates a potential relationship between the varied neuronal timescales observed across brain regions [62] and their capacity to manage corresponding delay ranges in temporal credit assignment during learning processes. The learning factor α may be mediated by neuromodulators, such as acetylcholine, noradrenaline, serotonin, and dopamine [44], which has been shown that they act as gatekeepers for synaptic plasticity, determining how synapses increase or decrease in strength [19]. This is consistent with the function of α, which controls the time range to assign error credits. Future work should explore how to adaptively parameterize the learning factor α based on network activity or neuromodulation. In this study, learning signals in ES-D-RTRL $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}$ are primarily generated by the backpropagation algorithm through deep layers. While it is effective, it is widely recognized as not being biologically plausible [34]. To enhance biological plausibility, alternative methods, such as random feedback alignment [33] and forward-forward algorithms [57], can be employed. Another promising approach is shallow learning [68]. Research shows that almost every cortical area communicates directly with subcortical areas like the brainstem, thalamus, and striatum, giving the brain a shallow architecture from their perspective, known as the shallow brain hypothesis [68]. In the context of our online learning algorithms, credit assignments through deep layers result in poor gradient approximation (Figure S15), and precise gradients with local online learning rules require a shadow network architecture, for example, each layer has a local loss function. Future research should investigate the role of subcortical structures in local loss optimization of large-scale brain networks.

Practitioners in neuroscience and artificial intelligence often ask: what unique benefits do spiking networks offer over traditional rate-based models? While energy efficiency is a well-known advantage of SNNs due to their discrete, event-driven nature [60], our study reveals an additional benefit in the realm of learning algorithms. Through both theoretical and empirical analyses, we demonstrate that gradient computation in SNNs can be effectively approximated using an online learning algorithm with a complexity of $\mathcal{O}(N)$ (Eqs. 4-6). This efficiency stems from the inherent properties of SNNs. Our findings suggest that SNNs have a distinct advantage in online learning, particularly for realizing powerful on-chip learning on neuromorphic chips. This positions them as a promising choice for developing advanced neural computation systems that can learn and adapt in real-time, ultimately bringing digital whole-brain simulations closer to reality.

7 Methods

7.1 Model Abstraction in *BrainScale*

BrainScale supports the online learning of models constructed based on the `AlignPre` and `AlignPost` projection principles (please refer to Section 2). These principles, previously utilized in our BrainPy simulator [75, 77] to create efficient and standardized SNN models, are now formally defined here. Our formulation covers neuronal dynamics (Methods 7.1.1), synaptic dynamics (Methods 7.1.2), `AlignPre` projection (Methods 7.1.3) and `AlignPost` projection (Methods 7.1.4). We also provide examples of SNN models employing this abstraction in Methods 7.2.

7.1.1 Neuronal dynamics

The dynamics of a neuron are regulated by a complex interplay of ionic channels, neurotransmitter receptors, and intracellular signaling cascades, which collectively govern the neuron's membrane potential and excitability [29]. Moreover, neuronal dynamics are influenced by synaptic inputs, and neuromodulatory substances that can modulate the neuron's responsiveness [29]. As a result of these intricate processes, individual neurons exhibit diverse and complex spiking behaviors, displaying highly variable firing patterns in response to identical step currents [27]. To capture this diversity, a range of spiking neuron models has been proposed, including detailed compartmental models, reduced compartmental models, and point neuron models [24].

In general, we denote the neuronal state as $\mathbf{v}$ and the occurrence of a neuronal spike as $\mathbf{z}$, with their dynamics governed by the following equations:

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = f_n(\mathbf{v}^{t-1}, t, \mathbf{I}^t), \tag{8}$$

$$\mathbf{z}^t = \mathcal{H}(\mathbf{v}^t - v_{\mathrm{th}}), \tag{9}$$

where $\mathbf{I}$ represents the external current, f_n typically denotes an element-wise function describing the evolution of each neuronal state, v_{th} is the threshold membrane potential for spike generation, and $\mathcal{H}$ represents the Heaviside function used to determine whether a spike is generated at the current time step. Note that we treat $\mathbf{z}$ as an intermediate variable rather than one of the hidden states.

One-dimensional point neuron model: The simplest approach for describing neuronal excitability involves one-dimensional dynamical systems, which consist of a single variable. An example of such a system is the leaky integrate-and-fire (LIF) model [2], whose dynamics is given by:

$$\tau \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -(\mathbf{v} - V_{\mathrm{rest}}) + \mathbf{I}, \tag{10}$$

where τ is the membrane time constant, and V_{rest} is the resting potential. Whenever the membrane potential reaches the spike threshold V_{th} (i.e., $\mathbf{v}_i(t) \geq V_{th}$), the neuron fires a spike, and the membrane potential is reset to the reset potential (i.e., $\mathbf{v}_i(t) \leftarrow V_{\mathrm{reset}}$). Using the exponential Euler integration, the LIF neuron can be discretized as:

$$\bar{\mathbf{v}}^{t-1} = \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\mathrm{reset}} - \mathbf{v}^{t-1}), \tag{11}$$

$$\mathbf{v}^t = e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\mathrm{rest}} + \mathbf{I}^t). \tag{12}$$

Due to its simplicity and computational efficiency, the LIF model is widely utilized in spiking network modeling but it cannot precisely model the subthreshold dynamics. To capture subthreshold fluctuations of the membrane potential more accurately, quadratic [32] or exponential [18] extensions of the LIF model can also be employed.

Two-dimensional point neuron model: However, one-dimensional models are limited in their ability to reproduce certain significant features of single neuron dynamics, such as adaptation and bursting. Models with at least two-dimensional states are required to faithfully replicate all the neurocomputational characteristics of neurons. Prominent examples include the adaptive exponential integrate-and-fire (AdEx) model [12] and the Izhikevich neuron model [26]. In this work, we are using the adaptive LIF (ALIF) model [5], whose dynamics is given by:

$$\tau_{\mathbf{a}} \frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} = -\mathbf{a}, \tag{13}$$

$$\tau \frac{d\mathbf{v}}{dt} = -\mathbf{v} + V_{\mathrm{rest}} + \mathbf{a} + \mathbf{I}, \tag{14}$$

where $\mathbf{a}$ denotes the internal adaption current and $\tau_{\mathbf{a}}$ is the time constant of the adaption current. When $\mathbf{v}^i$ of i-th neuron meets V_{th}, the model fires:

$$\mathbf{a}_i \leftarrow \mathbf{a}_i + A, \tag{15}$$

$$\mathbf{v}_i \leftarrow V_{\text{reset}}, \tag{16}$$

where A is the spike-triggered current.

Using the exponential Euler integration, the ALIF neuron can be discretized as:

$$\bar{\mathbf{v}}^{t-1} = \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\text{reset}} - \mathbf{v}^{t-1}), \tag{17}$$

$$\mathbf{a}^t = e^{-\Delta t/\tau_{\mathbf{a}}}\mathbf{a}^{t-1} + \mathbf{z}^{t-1}A, \tag{18}$$

$$\mathbf{v}^t = e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\text{rest}} + \mathbf{I}^t + \mathbf{a}^t). \tag{19}$$

Multi-compartment neuron model: Recently, dendritic mechanisms have been incorporated into simplified point-based neuron models [51] to account for important contributions of dendrites to neuronal integration and output, thereby enhancing the computational power of realistic neurons. These models employ phenomenological equations to describe dendritic compartments and spikes, and can be easily trained using gradient-based learning methods. Dendritic mechanisms operate across multiple timescales, ranging from a few milliseconds to hundreds of milliseconds, enabling complex computations such as coincidence detection, low-pass filtering, input segregation/amplification, parallel nonlinear processing, and logical operations [35, 72, 9, 45, 23, 67, 30, 72, 28, 54]. Recent SNN networks that incorporate similar dendritic mechanisms have demonstrated superior advantages in temporal integration [86] and long-term dependency learning [85].

7.1.2 Synaptic dynamics

Synaptic dynamics, on the other hand, describe the processes that occur at the synapses, which facilitate communication between neurons. Synaptic dynamics typically involve three processes: neurotransmitter release from the presynaptic neuron, binding of these neurotransmitters to receptors on the postsynaptic cell (which is further detailed in Section 7.1.3 and 7.1.4), and subsequent generation of postsynaptic excitatory or inhibitory currents.

1. Presynaptic neurotransmitter releasing. Firstly, synaptic kinetics are defined by presynaptic efficiency, such as the number of neurotransmitters released from the presynaptic cell [71, 56]. This phenomenon is known as short-term plasticity (STP), although it can be disregarded in modeling when the STP effect is weak. STP can take two forms: short-term facilitation (STF) and short-term depression (STD), which respectively increase and decrease synaptic efficiency with repeated presynaptic activity. In the *BrainScale* framework, the STP effect can be modeled using the `AlignPre` projection, where each neuron has a corresponding STP state.

STD model: The STD effect is modeled using a normalized variable $\mathbf{x}$ (where each element $0 \leq \mathbf{x}_i \leq 1$), representing the fraction of available resources after neurotransmitter depletion [70]:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \frac{1 - \mathbf{x}}{\tau_d} - U\mathbf{x}\delta(t - t^{\text{pre}}), \tag{20}$$

where U is the fraction of resources used per spike, τ_d is the time constant for recovery of synaptic vesicles, t^{pre} is the arrival time of the presynaptic spike, and $\delta(*)$ is the Dirac delta function representing spike occurrences. For numerical integration, the exponential Euler method can be used to accurately solve the STD model:

$$\mathbf{x}^t = e^{-\Delta t/\tau_d}\mathbf{x}^{t-1} + (1 - e^{-\Delta t/\tau_d}) - U\mathbf{x}^{t-1}\mathbf{z}^{\text{pre},t-t_d}, \tag{21}$$

where Δt is the numerical integration time step and t_d is the spike transmission delay.

STP model: The STF effect is modeled using a utilization parameter $\mathbf{u}$, representing the fraction of available resources ready for use (release probability). The dynamics of $\mathbf{u}$ and $\mathbf{x}$ are governed by the following equations:

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = -\frac{\mathbf{u}}{\tau_f} + U(1 - \mathbf{u}^-)\delta(t - t^{\text{pre}}), \tag{22}$$

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \frac{1 - \mathbf{x}}{\tau_d} - \mathbf{u}^+\mathbf{x}^-\delta(t - t^{\text{pre}}), \tag{23}$$

where $\mathbf{u}^-$ and $\mathbf{x}^-$ are the corresponding variables just before the arrival of the spike, and $\mathbf{u}^+$ refers to the moment just after the spike. In the parameter regime of $\tau_d \gg \tau_f$ and large U, an initial spike leads to a significant drop in $\mathbf{x}$ that takes a long time to recover, resulting in STD dominance. In the regime of $\tau_f \gg \tau_d$ and small U, the synaptic efficacy gradually increases with spikes, leading to STF dominance. Using the exponential Euler integration method, the STP model can be discretized as follows:

$$\mathbf{u}^t = e^{-\Delta t/\tau_f}\mathbf{u}^{t-1} + U(1-\mathbf{u}^{t-1})\mathbf{z}^{\mathrm{pre},t-t_d}, \tag{24}$$

$$\mathbf{x}^t = e^{-\Delta t/\tau_d}\mathbf{x}^{t-1} + (1-e^{-\Delta t/\tau_d}) - \mathbf{u}^t\mathbf{x}^{t-1}\mathbf{z}^{\mathrm{pre},t-t_d}. \tag{25}$$

2. Postsynaptic neurotransmitter binding. Following the neurotransmitter release, synaptic efficiency is commonly characterized by the postsynaptic neurotransmitter binding ratio. This binding is denoted as the synaptic states $\mathbf{g}$, and their dynamics are governed by the equation:

$$\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}t} = f_g(\mathbf{g}, t), \tag{26}$$

where f_g represents an element-wise function that specifies the evolution of each synaptic state based on neurotransmitter binding.

The specific formulation for modeling postsynaptic neurotransmitter binding is not provided in this section since the modeling approaches differ between the `AlignPre` and `AlignPost` projections in *BrainScale*. For detailed information regarding neurotransmitter binding, please refer to Methods 7.1.3 and Methods 7.1.4.

3. Postsynaptic current generation. Finally, once the postsynaptic conductance is obtained, the changes in current at the postsynaptic site can be computed using Ohm's law. The postsynaptic conductance induced by synaptic connections is denoted as $\mathbf{g}^{\mathrm{post},t}$. Please refer to Methods 7.1.3 and Methods 7.1.4 for detailed information on how it is calculated in `AlignPre` and `AlignPost` projections, respectively.

***COBA**:* Based on Ohm's law, the current generated in the postsynaptic cells is expressed as the conductance-based output (COBA):

$$\mathbf{I}^{\mathrm{post},t} = \mathbf{g}^{\mathrm{post},t} \circ (E - \mathbf{v}^{\mathrm{post},t}), \tag{27}$$

where E represents the reversal potential.

***CUBA**:* However, since the membrane voltage typically fluctuates around the resting membrane potential v_{rest}, the above equation can be simplified as:

$$\mathbf{I}^{\mathrm{post},t} = \mathbf{g}^{\mathrm{post},t}(E - v_{\mathrm{rest}}) \propto \mathbf{g}^{\mathrm{post},t}, \tag{28}$$

where $E - v_{\mathrm{rest}}$ is a constant that can be absorbed into the synaptic weight $\boldsymbol{\theta}$. This model is referred to as current-based output (CUBA).

***NMDA**:* In the case of NMDA synapses, the current flux is also influenced by magnesium. Given the synaptic conductance, the NMDA synapses generate the postsynaptic current using the following equation:

$$\mathbf{I}^{\mathrm{post},t} = \mathbf{g}^{\mathrm{post},t} \circ (E - \mathbf{v}^{\mathrm{post},t}) \circ g_\infty(\mathbf{v}^{\mathrm{post},t}), \tag{29}$$

where the fraction of channels g_∞ not blocked by magnesium can be fitted as $g_\infty(\mathbf{v}) = (1 + e^{-\alpha\mathbf{v}\frac{[\mathrm{Mg}^{2+}]}{\beta}})^{-1}$. Here, $[\mathrm{Mg}^{2+}]$ represents the extracellular magnesium concentration.

7.1.3 `AlignPre` synaptic projections

In the `AlignPre` projection (Figure 2C), the synaptic state $\mathbf{g}$ represents the changes in conductance on postsynaptic neurons induced by each presynaptic neuronal spike. Therefore, $\mathbf{g} \in \mathbb{R}^n$ has dimensions aligned with the presynaptic neurons. Two models are introduced below: the Alpha synapse and Markov synapse models.

***Alpha model**:* The Alpha synapse is a commonly used model, described analytically as $\mathbf{g}^t = \frac{t-t^{\mathrm{pre}}}{\tau_{\mathbf{g}}} \exp\left(-\frac{t-t^{\mathrm{pre}}}{\tau_{\mathbf{g}}}\right)$. However, this equation is challenging to implement directly. It can be

transformed into differential forms for numerical integration:

$$\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}t} = -\frac{\mathbf{g}}{\tau_{\mathbf{g}}} + \frac{\mathbf{s}}{\tau_{\mathbf{g}}}, \tag{30}$$

$$\frac{\mathrm{d}\mathbf{s}}{\mathrm{d}t} = -\frac{\mathbf{s}}{\tau_{\mathbf{g}}} + \delta\left(t - t^{\mathrm{pre}}\right), \tag{31}$$

where $\mathbf{s}$ is an intermediate variable. By employing the exponential Euler integration method, the Alpha synapse model can be discretized as follows:

$$\mathbf{s}^t = e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{s}^{t-1} + \mathbf{z}^{\mathrm{pre},t-t_d}, \tag{32}$$

$$\mathbf{g}^t = e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{g}^{t-1} + (1 - e^{-\Delta t/\tau_{\mathbf{g}}})\mathbf{s}^t. \tag{33}$$

Synaptic interaction: The aforementioned `AlignPre` synaptic models describe how conductance changes on postsynaptic neurons are driven by each presynaptic neuron. In order to combine the effects of all presynaptic connections onto a single postsynaptic neuron, we need to compute the synaptic interaction. Typically, a linear transformation is used to calculate the aggregated postsynaptic conductance:

$$\mathbf{g}^{\mathrm{post},t} = \boldsymbol{\theta} \circledast \mathbf{g}^t, \tag{34}$$

Here, $\boldsymbol{\theta} \in \mathbb{R}^{n\times m}$ represents the synaptic connectivity, which can be a dense or sparse matrix. It is worth noting that the transformation $\boldsymbol{\theta}$ can also be implemented as a convolution or other operations commonly used in deep learning.

By incorporating the STP effect, the computation of the postsynaptic conductance can be modified as follows:

$$\mathbf{g}^{\mathrm{post},t} = \begin{cases} \boldsymbol{\theta} \circledast (\mathbf{u}^t \circ \mathbf{x}^t \circ \mathbf{g}^t), & \text{for STP,} \\ \boldsymbol{\theta} \circledast (\mathbf{x}^t \circ \mathbf{g}^t), & \text{for STD.} \end{cases} \tag{35}$$

In the case of STP, the conductance is computed by element-wise multiplication of vectors $\mathbf{u}^t$, $\mathbf{x}^t$, and $\mathbf{g}^t$, followed by the linear transformation $\boldsymbol{\theta}$. On the other hand, in the case of STD, only vectors $\mathbf{x}^t$ and $\mathbf{g}^t$ are element-wise multiplied before applying the transformation $\boldsymbol{\theta}$.

7.1.4 `AlignPost` synaptic projections

In the `AlignPost` projection (Figure 2D), the synaptic state $\mathbf{g} \in \mathbb{R}^m$ represents the summarized conductance effect on the specific postsynaptic site. Unlike `AlignPre` projections, `AlignPost` models initially calculate the aggregated conductance changes based on the connected presynaptic inputs (refer to the next paragraph "Synaptic interaction"). Subsequently, the synaptic conductance evolves in response to the summarized conductance inputs using exponential-family synapse models. Notably, the `AlignPost` projection can only apply to exponential-family synapse models.

Synaptic interaction: In contrast to the `AlignPre` projection, the synaptic interactions in `AlignPost` projections should be first computed. It transforms the presynaptic spikes $\mathbf{z}^{\mathrm{pre},t-t_d}$ into postsynaptic conductance changes $\Delta\mathbf{g}^{\mathrm{post}}$ using the following equation:

$$\Delta\mathbf{g}^{\mathrm{post},t} = \boldsymbol{\theta} \circledast \mathbf{z}^{\mathrm{pre},t-t_d}. \tag{36}$$

Since $\mathbf{z}$ represents binary spikes, this transformation can be efficiently performed using event-driven operators provided in the BrainPy simulator [75, 77]. However, when the STP effect is considered (see Methods 7.1.2), `AlignPost` projections calculate postsynaptic conductance changes as follows:

$$\Delta\mathbf{g}^{\mathrm{post},t} = \begin{cases} \boldsymbol{\theta} \circledast (\mathbf{u}^t \circ \mathbf{x}^t \circ \mathbf{z}^{\mathrm{pre},t-t_d}), & \text{for STP,} \\ \boldsymbol{\theta} \circledast (\mathbf{x}^t \circ \mathbf{z}^{\mathrm{pre},t-t_d}), & \text{for STD,} \end{cases} \tag{37}$$

where $\mathbf{u}$ and $\mathbf{x}$ are variables representing short-term facilitation and depression, respectively.

Once the postsynaptic conductance changes are computed, the postsynaptic conductance $\mathbf{g}^{\mathrm{post},t}$ can be updated according to the following models.

Exponential model: Assuming the binding of neurotransmitters is instantaneous, the single exponential model only models the decay phase of a neurotransmitter binding as $\mathbf{g}^{\mathrm{post},t} = \exp\left(-\left(t - t^{\mathrm{pre}}\right)/\tau_{\mathbf{g}}\right)$, where $\tau_{\mathbf{g}}$ is the decay time constant. This equation can be expressed in a differential equation form:

$$\frac{\mathrm{d}\mathbf{g}^{\mathrm{post}}}{\mathrm{d}t} = -\frac{\mathbf{g}^{\mathrm{post}}}{\tau_{\mathbf{g}}} + \sum_k \delta(t - t_k^{\mathrm{sp}}). \tag{38}$$

Using the exponential Euler integration method, the exponential synapse model can be discretized as:

$$\mathbf{g}^{\mathrm{post},t} = e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{g}^{\mathrm{post},t-1} + \Delta\mathbf{g}^{\mathrm{post},t}. \tag{39}$$

Double exponential model: Considering both the decay and rice phases, the double exponential model reproduces the behavior of the neurotransmitter binding well: $\mathbf{g}^{\mathrm{post},t} = B\left(\exp\left(-\frac{t-t^{\mathrm{pre}}}{\tau_d}\right) - \exp\left(-\frac{t-t^{\mathrm{pre}}}{\tau_r}\right)\right)$, where $\tau_{\mathbf{g}}$ is the decay time constant, τ_r is the rise time constant, and $B = \frac{\tau_{\mathbf{g}}}{\tau_{\mathbf{g}}-\tau_r}\left(\frac{\tau_r}{\tau_{\mathbf{g}}}\right)^{\frac{\tau_r}{\tau_r-\tau_{\mathbf{g}}}}$ is the normalizing constant so that the maximal value of $\mathbf{g}^{\mathrm{post}}$ is 1 for a single spike. The double exponential model can be expressed as:

$$\mathbf{g}^{\mathrm{post},t} = B\left(\mathbf{s}_d^t - \mathbf{s}_r^t\right), \tag{40}$$

$$\mathbf{s}_d^t = e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{s}_d^{t-1} + \Delta\mathbf{g}^{\mathrm{post},t}, \tag{41}$$

$$\mathbf{s}_r^t = e^{-\Delta t/\tau_r}\mathbf{s}_r^{t-1} + \Delta\mathbf{g}^{\mathrm{post},t}. \tag{42}$$

7.2 Spiking Networks Evaluated in this Work

In this section, we present the spiking networks evaluated in this study. We provide a detailed description of these networks through their discretized equations, which form the foundation of our analysis. A key focus is placed on the neuronal interaction (see Methods 7.3), highlighted using the notation `neuronal interaction` with an under bracket. Other element-wise operations in the equations describe the intrinsic dynamics of individual neurons and synapses within the network. Additionally, we explicitly point out the concrete implementation of the hidden state $\mathbf{h}^t$ in Eq. 1, its diagonal elements of $\mathbf{D}^t$ and the hidden-to-weight diagonal matrix $\mathbf{D}_f^t$ in Eq. 3-6 for each network. Note that all partial derivatives in this section do not flow through the recurrent spikes $\mathbf{z}^{t-1}$, and mathematical notations are consistent with Methods 7.1.

The network with LIF neurons and Delta synapses. The network comprising LIF neurons interconnected via Delta synapses represents a foundational model in computational neuroscience and neuromorphic engineering. Its discretized dynamics can be represented as:

$$\begin{aligned}\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\mathrm{reset}} - \mathbf{v}^{t-1}),\\ \mathbf{v}^t &= e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\mathrm{rest}} + \underbrace{\boldsymbol{\theta}^{\mathrm{rec}} \circledast \mathbf{z}^{t-1} + \boldsymbol{\theta}^{\mathrm{in}} \circledast \mathbf{z}^{\mathrm{in},t}}_{\text{neuronal interaction}}),\end{aligned} \tag{43}$$

where $\mathbf{z}^{\mathrm{in},t} \in \mathbb{R}^I$ is the input spike at the time t, $\boldsymbol{\theta}^{\mathrm{rec}} \in \mathbb{R}^{H\times H}$ is the recurrent weight, and $\boldsymbol{\theta}^{\mathrm{in}} \in \mathbb{R}^{I\times H}$ is the input weight. The hidden state $\mathbf{h}^t$ in Eq. 1 corresponds to $\mathbf{v}^t$ in this network, $\mathbf{D}_{ii}^t = \partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1}$, and $\mathbf{D}_{f,ii}^t = \partial\mathbf{v}_i^t/\partial\mathbf{I}_i^t$.

The network with ALIF neurons and Delta synapses. Its discretized dynamics is given by:

$$\begin{aligned}\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\mathrm{reset}} - \mathbf{v}^{t-1}),\\ \mathbf{a}^t &= e^{-\Delta t/\tau_{\mathbf{a}}}\mathbf{a}^{t-1} + \mathbf{z}^{t-1}A,\\ \mathbf{v}^t &= e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\mathrm{rest}} + \mathbf{a}^t + \underbrace{\boldsymbol{\theta}^{\mathrm{rec}} \circledast \mathbf{z}^{t-1} + \boldsymbol{\theta}^{\mathrm{in}} \circledast \mathbf{z}^{\mathrm{in},t}}_{\text{neuronal interaction}}),\end{aligned} \tag{44}$$

where $\mathbf{a}^t \in \mathbb{R}^H$ is the internal adaption current, $\mathbf{h}_i^t = \begin{pmatrix}\mathbf{v}_i^t\\ \mathbf{a}_i^t\end{pmatrix}$, $\mathbf{D}_{ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{a}_i^{t-1}\\ \partial\mathbf{a}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{a}_i^{t-1}\end{pmatrix}$, and $\mathbf{D}_{f,ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{I}_i^{t-1}\\ \partial\mathbf{a}_i^t/\partial\mathbf{I}_i^{t-1}\end{pmatrix}$.

The network with LIF neurons and current-based Exponential synapses. The LIF neuron networks with current-based exponential synapses constitute another foundational model in computational neuroscience. Its discretized dynamics can be represented as:

$$\begin{aligned}\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\text{reset}} - \mathbf{v}^{t-1}),\\ \mathbf{g}^{t} &= e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{g}^{t-1} + \underbrace{\boldsymbol{\theta}^{\text{rec}} \circledast \mathbf{z}^{t-1} + \boldsymbol{\theta}^{\text{in}} \circledast \mathbf{z}^{\text{in},t}}_{\text{neuronal interaction}},\\ \mathbf{v}^{t} &= e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\text{rest}} + \mathbf{g}^{t}),\end{aligned} \tag{45}$$

where the synaptic current $\mathbf{g}^t \in \mathbb{R}^H$ is modeled using the exponential synapse with the CUBA output. In this network, $\mathbf{h}_i^t = \begin{pmatrix}\mathbf{v}_i^t\\ \mathbf{g}_i^t\end{pmatrix}$, $\mathbf{D}_{ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{g}_i^{t-1}\\ \partial\mathbf{g}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{g}_i^{t-1}\end{pmatrix}$, and $\mathbf{D}_{f,ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{I}_i^{t-1}\\ \partial\mathbf{g}_i^t/\partial\mathbf{I}_i^{t-1}\end{pmatrix}$.

The network with ALIF neurons and current-based Exponential synapses. The dynamics of the network can be discretized as:

$$\begin{aligned}\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\text{reset}} - \mathbf{v}^{t-1}),\\ \mathbf{g}^{t} &= e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{g}^{t-1} + \underbrace{\boldsymbol{\theta}^{\text{rec}} \circledast \mathbf{z}^{t-1} + \boldsymbol{\theta}^{\text{in}} \circledast \mathbf{z}^{\text{in},t}}_{\text{neuronal interaction}},\\ \mathbf{a}^{t} &= e^{-\Delta t/\tau_{\mathbf{a}}}\mathbf{a}^{t-1} + \mathbf{z}^{t-1}A,\\ \mathbf{v}^{t} &= e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\text{rest}} + \mathbf{a}^{t} + \mathbf{g}^{t}),\end{aligned} \tag{46}$$

where $\mathbf{h}_i^t = \begin{pmatrix}\mathbf{v}_i^t\\ \mathbf{a}_i^t\\ \mathbf{g}_i^t\end{pmatrix}$, $\mathbf{D}_{ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{g}_i^{t-1}\\ \partial\mathbf{a}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{g}_i^{t-1}\\ \partial\mathbf{g}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{g}_i^{t-1}\end{pmatrix}$, and $\mathbf{D}_{f,ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{I}_i^{t-1}\\ \partial\mathbf{a}_i^t/\partial\mathbf{I}_i^{t-1}\\ \partial\mathbf{g}_i^t/\partial\mathbf{I}_i^{t-1}\end{pmatrix}$

The network with ALIF neurons and conductance-based Exponential synapses. The dynamics of the network can be discretized as:

$$\begin{aligned}\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\text{reset}} - \mathbf{v}^{t-1}),\\ \mathbf{g}^{t} &= e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{g}^{t-1} + \underbrace{\boldsymbol{\theta}^{\text{rec}} \circledast \mathbf{z}^{t-1} + \boldsymbol{\theta}^{\text{in}} \circledast \mathbf{z}^{\text{in},t}}_{\text{neuronal interaction}},\\ \mathbf{a}^{t} &= e^{-\Delta t/\tau_{\mathbf{a}}}\mathbf{a}^{t-1} + \mathbf{z}^{t-1}A,\\ \mathbf{v}^{t} &= e^{-\Delta t(1+\mathbf{g}^t)/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t(1+\mathbf{g}^t)/\tau})(V_{\text{rest}} + \mathbf{a}^{t} + E \circ \mathbf{g}^{t}),\end{aligned} \tag{47}$$

where $\mathbf{h}_i^t = \begin{pmatrix}\mathbf{v}_i^t\\ \mathbf{a}_i^t\\ \mathbf{g}_i^t\end{pmatrix}$, $\mathbf{D}_{ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{g}_i^{t-1}\\ \partial\mathbf{a}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{g}_i^{t-1}\\ \partial\mathbf{g}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{g}_i^{t-1}\end{pmatrix}$, and $\mathbf{D}_{ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{I}_i^{t-1}\\ \partial\mathbf{a}_i^t/\partial\mathbf{I}_i^{t-1}\\ \partial\mathbf{g}_i^t/\partial\mathbf{I}_i^{t-1}\end{pmatrix}$.

The network with LIF neurons, STD, and current-based Exponential synapses. The dynamics of the network can be discretized as:

$$\begin{aligned}\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\text{reset}} - \mathbf{v}^{t-1}),\\ \mathbf{x}^{t} &= e^{-\Delta t/\tau_d}\mathbf{x}^{t-1} + (1 - e^{-\Delta t/\tau_d}) - U\mathbf{x}^{t-1}\mathbf{z}^{t-1},\\ \mathbf{g}^{t} &= e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{g}^{t-1} + \underbrace{\boldsymbol{\theta}^{\text{rec}} \circledast (\mathbf{x}^{t} \circ \mathbf{z}^{t-1}) + \boldsymbol{\theta}^{\text{in}} \circledast \mathbf{x}\mathbf{z}^{\text{in},t}}_{\text{neuronal interaction}},\\ \mathbf{v}^{t} &= e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\text{rest}} + \mathbf{g}^{t}),\end{aligned} \tag{48}$$

where $\mathbf{x}^t \in \mathbb{R}^H$ is the short-term depression variable for the recurrent spikes, and $\mathbf{x}\mathbf{z}^{\text{in},t} \in \mathbb{R}^I$ is the short-term depression variable for the input spikes, $\mathbf{h}_i^t = \begin{pmatrix}\mathbf{v}_i^t\\ \mathbf{x}_i^t\\ \mathbf{g}_i^t\end{pmatrix}$, $\mathbf{D}_{ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{g}_i^{t-1}\\ \partial\mathbf{x}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{g}_i^{t-1}\\ \partial\mathbf{g}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{g}_i^{t-1}\end{pmatrix}$, and $\mathbf{D}_{f,ii}^t = \begin{pmatrix}\partial\mathbf{v}_i^t/\partial\mathbf{I}_i^{t-1}\\ \partial\mathbf{x}_i^t/\partial\mathbf{I}_i^{t-1}\\ \partial\mathbf{g}_i^t/\partial\mathbf{I}_i^{t-1}\end{pmatrix}$

The network with ALIF neurons, STD, and current-based Exponential synapses. The dynamics of the network can be discretized as:

$$
\begin{aligned}
\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\text{reset}} - \mathbf{v}^{t-1}), \\
\mathbf{x}^{t} &= e^{-\Delta t/\tau_d}\mathbf{x}^{t-1} + (1 - e^{-\Delta t/\tau_d}) - U\mathbf{x}^{t-1}\mathbf{z}^{t-1}, \\
\mathbf{g}^{t} &= e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{g}^{t-1} + \underbrace{\boldsymbol{\theta}^{\text{rec}} \circledast (\mathbf{x}^{t} \circ \mathbf{z}^{t-1}) + \boldsymbol{\theta}^{\text{in}} \circledast \mathbf{x}\mathbf{z}^{\text{in},t}}_{\text{neuronal interaction}}, \\
\mathbf{a}^{t} &= e^{-\Delta t/\tau_{\mathbf{a}}}\mathbf{a}^{t-1} + \mathbf{z}^{t-1}A, \\
\mathbf{v}^{t} &= e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\text{rest}} + \mathbf{a}^{t} + \mathbf{g}^{t}),
\end{aligned}
\tag{49}
$$

where $\mathbf{h}_i^t = \begin{pmatrix} \mathbf{v}_i^t \\ \mathbf{x}_i^t, \\ \mathbf{a}_i^t, \\ \mathbf{g}_i^t \end{pmatrix}$, $\mathbf{D}_{ii}^t = \begin{pmatrix} \partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{x}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{a}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{g}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{g}_i^{t-1} \end{pmatrix}$, and $\mathbf{D}_{f,ii}^t = \begin{pmatrix} \partial\mathbf{v}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{x}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{a}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{g}_i^t/\partial\mathbf{I}_i^{t-1} \end{pmatrix}$.

The network with LIF neurons, STP, and current-based Exponential synapses. The dynamics of the network can be discretized as:

$$
\begin{aligned}
\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\text{reset}} - \mathbf{v}^{t-1}), \\
\mathbf{u}^{t} &= e^{-\Delta t/\tau_f}\mathbf{u}^{t-1} + U(1 - \mathbf{u}^{t-1})\mathbf{z}^{t-1}, \\
\mathbf{x}^{t} &= e^{-\Delta t/\tau_d}\mathbf{x}^{t-1} + (1 - e^{-\Delta t/\tau_d}) - \mathbf{u}^{t}\mathbf{x}^{t-1}\mathbf{z}^{t-1}, \\
\mathbf{g}^{t} &= e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{g}^{t-1} + \underbrace{\boldsymbol{\theta}^{\text{rec}} \circledast (\mathbf{u}^{t} \circ \mathbf{x}^{t} \circ \mathbf{z}^{t-1}) + \boldsymbol{\theta}^{\text{in}} \circledast \mathbf{u}\mathbf{x}\mathbf{z}^{\text{in},t}}_{\text{neuronal interaction}}, \\
\mathbf{v}^{t} &= e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\text{rest}} + \mathbf{g}^{t}),
\end{aligned}
\tag{50}
$$

where $\mathbf{u}^t, \mathbf{x}^t \in \mathbb{R}^H$ are short-term plasticity variables for the recurrent spikes, and $\mathbf{u}\mathbf{x}\mathbf{z}^{\text{in},t} \in \mathbb{R}^I$ is the short-term plasticity variable for the input spikes, $\mathbf{h}_i^t = \begin{pmatrix} \mathbf{v}_i^t \\ \mathbf{u}_i^t \\ \mathbf{x}_i^t \\ \mathbf{g}_i^t \end{pmatrix}$, $\mathbf{D}_{ii}^t = \begin{pmatrix} \partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{u}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{u}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{u}_i^t/\partial\mathbf{u}_i^{t-1} & \partial\mathbf{u}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{u}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{x}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{u}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{g}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{u}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{g}_i^{t-1} \end{pmatrix}$, and $\mathbf{D}_{f,ii}^t = \begin{pmatrix} \partial\mathbf{v}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{u}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{x}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{g}_i^t/\partial\mathbf{I}_i^{t-1} \end{pmatrix}$.

The network with ALIF neurons, STP, and current-based Exponential synapses. The dynamics of the network can be discretized as:

$$
\begin{aligned}
\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\text{reset}} - \mathbf{v}^{t-1}), \\
\mathbf{u}^{t} &= e^{-\Delta t/\tau_f}\mathbf{u}^{t-1} + U(1 - \mathbf{u}^{t-1})\mathbf{z}^{t-1}, \\
\mathbf{x}^{t} &= e^{-\Delta t/\tau_d}\mathbf{x}^{t-1} + (1 - e^{-\Delta t/\tau_d}) - \mathbf{u}^{t}\mathbf{x}^{t-1}\mathbf{z}^{t-1}, \\
\mathbf{g}^{t} &= e^{-\Delta t/\tau_{\mathbf{g}}}\mathbf{g}^{t-1} + \underbrace{\boldsymbol{\theta}^{\text{rec}} \circledast (\mathbf{u}^{t} \circ \mathbf{x}^{t} \circ \mathbf{z}^{t-1}) + \boldsymbol{\theta}^{\text{in}} \circledast \mathbf{u}\mathbf{x}\mathbf{z}^{\text{in},t}}_{\text{neuronal interaction}}, \\
\mathbf{a}^{t} &= e^{-\Delta t/\tau_{\mathbf{a}}}\mathbf{a}^{t-1} + \mathbf{z}^{t-1}A, \\
\mathbf{v}^{t} &= e^{-\Delta t/\tau}\bar{\mathbf{v}}^{t-1} + (1 - e^{-\Delta t/\tau})(V_{\text{rest}} + \mathbf{a}^{t} + \mathbf{g}^{t}),
\end{aligned}
\tag{51}
$$

where $\mathbf{h}_i^t = \begin{pmatrix} \mathbf{v}_i^t \\ \mathbf{u}_i^t \\ \mathbf{x}_i^t \\ \mathbf{a}_i^t \\ \mathbf{g}_i^t \end{pmatrix}$, $\mathbf{D}_{ii}^t = \begin{pmatrix} \partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{u}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{u}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{u}_i^t/\partial\mathbf{u}_i^{t-1} & \partial\mathbf{u}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{u}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{u}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{x}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{u}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{x}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{a}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{u}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{g}_i^{t-1} \\ \partial\mathbf{g}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{u}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{x}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{g}_i^t/\partial\mathbf{g}_i^{t-1} \end{pmatrix}$, and $\mathbf{D}_{f,ii}^t = \begin{pmatrix} \partial\mathbf{v}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{u}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{x}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{a}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{g}_i^t/\partial\mathbf{I}_i^{t-1} \end{pmatrix}$.

The E/I network with ALIF neurons and conductance-based Exponential synapses. Its discretized dynamics is given by:

$$
\begin{aligned}
\bar{\mathbf{v}}^{t-1} &= \mathbf{v}^{t-1} + \mathbf{z}^{t-1}(V_{\text{reset}} - \mathbf{v}^{t-1}), \\
\mathbf{g}_{\text{e}}^t &= e^{-\Delta t/\tau_{\text{e}}}\mathbf{g}_{\text{e}}^{t-1} + \underbrace{|\boldsymbol{\theta}_{\text{e}}^{\text{rec}}| \circledast \mathbf{z}_{\text{e}}^{t-1} + |\boldsymbol{\theta}^{\text{in}}| \circledast \mathbf{z}^{\text{in},t}}_{\text{neuronal interaction}}, \\
\mathbf{g}_{\text{i}}^t &= e^{-\Delta t/\tau_{\text{i}}}\mathbf{g}_{\text{i}}^{t-1} - \underbrace{|\boldsymbol{\theta}_{\text{i}}^{\text{rec}}| \circledast \mathbf{z}_{\text{i}}^{t-1}}_{\text{neuronal interaction}}, \\
\mathbf{d}^t &= \exp(-\Delta t(1 + \mathbf{g}_{\text{e}}^t + \mathbf{g}_{\text{i}}^t)/\tau), \\
\mathbf{a}^t &= e^{-\Delta t/\tau_{\mathbf{a}}}\mathbf{a}^{t-1} + \mathbf{z}^{t-1}A, \\
\mathbf{v}^t &= \mathbf{d}^t\bar{\mathbf{v}}^{t-1} + (1 - \mathbf{d}^t)(V_{\text{rest}} + \mathbf{a}^t + E_{\text{e}} \circ \mathbf{g}_{\text{e}}^t + E_{\text{i}} \circ \mathbf{g}_{\text{i}}^t),
\end{aligned}
\tag{52}
$$

where $\tau_{\text{e/i}}$ are time constants of excitatory/inhibitory synapses, $E_{\text{e/i}}$ are the excitatory/inhibitory reversal potentials, $\mathbf{g}_{\text{e/i}} \in \mathbb{R}^H$ represent the excitatory/inhibitory conductance, $\mathbf{z}_{\text{e}} \in \mathbb{R}^{H_{\text{e}}}$ denotes the spikes of excitatory neurons, $\mathbf{z}_{\text{i}} \in \mathbb{R}^{H_{\text{i}}}$ denotes the spikes of inhibitory neurons, $\boldsymbol{\theta}_{\text{e/i}}^{\text{rec}}$ are recurrent excitatory/inhibitory weights, $|*|$ denotes the absolute value, and $\mathbf{d}^t$ is an intermediate variable for membrane potential decay. In this network, the hidden state $\mathbf{h}_i^t = \begin{pmatrix} \mathbf{v}_i^t \\ \mathbf{a}_i^t \\ \mathbf{g}_{\text{e},i}^t \\ \mathbf{g}_{\text{i},i}^t \end{pmatrix}$, $\mathbf{D}_{ii}^t = \begin{pmatrix} \partial\mathbf{v}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{g}_{\text{e},i}^{t-1} & \partial\mathbf{v}_i^t/\partial\mathbf{g}_{\text{i},i}^{t-1} \\ \partial\mathbf{a}_i^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{g}_{\text{e},i}^{t-1} & \partial\mathbf{a}_i^t/\partial\mathbf{g}_{\text{i},i}^{t-1} \\ \partial\mathbf{g}_{\text{e},i}^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{g}_{\text{e},i}^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{g}_{\text{e},i}^t/\partial\mathbf{g}_{\text{e},i}^{t-1} & \partial\mathbf{g}_{\text{e},i}^t/\partial\mathbf{g}_{\text{i},i}^{t-1} \\ \partial\mathbf{g}_{\text{i},i}^t/\partial\mathbf{v}_i^{t-1} & \partial\mathbf{g}_{\text{i},i}^t/\partial\mathbf{a}_i^{t-1} & \partial\mathbf{g}_{\text{i},i}^t/\partial\mathbf{g}_{\text{e},i}^{t-1} & \partial\mathbf{g}_{\text{i},i}^t/\partial\mathbf{g}_{\text{i},i}^{t-1} \end{pmatrix}$, and $\mathbf{D}_{f,ii}^t = \begin{pmatrix} \partial\mathbf{v}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{a}_i^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{g}_{\text{e},i}^t/\partial\mathbf{I}_i^{t-1} \\ \partial\mathbf{g}_{\text{i},i}^t/\partial\mathbf{I}_i^{t-1} \end{pmatrix}$.

7.3 SNN Inherent Properties

This section offers a formal and detailed description of the intrinsic properties of SNNs. While some content overlaps with Section 3, this segment provides a more comprehensive explanation and employs rigorous mathematical formalism to elucidate the concepts.

Property 1: Intrinsic neuronal dynamics dominate the hidden Jacobian. A typical SNN model exhibits two types of recurrent connections. The first type comprises recurrent spiking connections that propagate action potentials between neurons (see $\circledast$ in Eq. 1 and examples in Methods 7.2). The second type consists of intrinsic neuronal dynamics, such as leaky terms in neuronal and synaptic processes (see f Eq. 1 and examples in Methods 7.2). As a result, the hidden Jacobian is composed of two parts: $\frac{\partial\mathbf{h}^t}{\partial\mathbf{h}^{t-1}} = \mathbf{D}^t + \mathbf{J}^t$, where $\mathbf{D}^t \in \mathbb{R}^{H\times H}$ is a diagonal matrix[1] resulted from the element-wise intrinsic dynamics and $\mathbf{J}^t$ is the square matrix derived from the recurrent spiking connections.

A defining characteristic of SNNs is their sparse activation patterns, with neurons firing at relatively low frequencies. Empirical studies have shown that biological neurons typically emit between 0.2 and 5 spikes per second [4], with only 1-2% of neurons in a cortical network active at any given moment

[1] $\mathbf{D}^t$ is actually a block diagonal matrix of dimensions $\mathbb{R}^{Hd\times Hd}$, consisting of H-by-H blocks. Each diagonal block, $\mathbf{D}_{ii}^t$, is a d-by-d square matrix, where d is the number of state variables aligned to neuron i. For examples, refer to Methods 7.2. Without loss of generality, all subsequent derivations assume $d = 1$, making $\mathbf{D}^t$ a diagonal matrix of dimensions $\mathbb{R}^{H\times H}$.

[47, 31]. This fact leads to the hypothesis that $\mathbf{J}^t$ exerts limited influence and can be ignored in the hidden Jacobian, since $\mathbf{J}^t$ is highly sparse in both space and time. Earlier practices, such as E-prop [6] and RFLO [43], have also demonstrated that neglecting the gradient flow through recurrent spiking connections in specific neuronal dynamics does not significantly hinder learning performance. To further validate this hypothesis, we conducted empirical evaluations across various SNN architectures with different neuronal and synaptic dynamics (Methods 7.2). Our findings consistently indicated that the hidden Jacobian of most SNN models is predominantly shaped by the diagonal Jacobian $\mathbf{D}^t$ induced by intrinsic dynamics. The diagonal approximation demonstrates a remarkably high cosine similarity with the full hidden Jacobian (see Figure 3A). Notably, under biologically plausible firing regimes (fraction of spiking neurons < 10%), the cosine similarity exceeds 99%.

Proposition 7.1. *In sparsely activated SNN models, the hidden Jacobian $\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}}$ is dominated by the diagonal Jacobian $\mathbf{D}^t$ induced by intrinsic neural dynamics. This can be expressed as:*

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} \approx \mathbf{D}^t. \tag{53}$$

Property 2: Physical connections yield decomposable hidden-to-weight Jacobian. Biological SNNs are characterized by physically constrained connections, reflecting the actual synaptic formations between pre- and postsynaptic neurons. Specifically, in `AlignPre` and `AlignPost` abstractions, this structural property can be mathematically expressed as a parameterized transformation from inputs to hidden vectors, followed by the element-wise activation applied to the resulting hidden vectors (see Eq. 1). Such mathematical operation leads to an important implication: the hidden-to-weight Jacobian $\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t}$, which quantifies the sensitivity of the network's output to changes in synaptic weights, can be decomposed into a Kronecker product of a vector and a diagonal matrix (see Theorem 7.2 and its proof in Methods 7.4; this observation has also been illustrated in [42] for rate-based recurrent neural networks (RNNs)).

Theorem 7.2. *Given Eq. 1, let $\mathbf{I}^t = \boldsymbol{\theta} \circledast \mathbf{x}^t$, then the gradient of $\mathbf{h}^t$ with respect to $\boldsymbol{\theta}$ at time t is*

$$\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t} = \mathbf{D}_f^t \otimes \mathbf{x}^t, \tag{54}$$

where $\mathbf{D}_f^t$ is a diagonal matrix[2] with $\frac{\partial \mathbf{h}_i^t}{\partial \mathbf{I}_i^t}$ as its diagonal entry $\mathbf{D}_{f,ii}^t$, and $\otimes$ is the Kronecker product.

Property 3: Sign-preserving neuronal outputs induce low-rank structure. Another unique characteristic of SNN models is that the outputs of each neuron, corresponding to inputs $\mathbf{x}^t$, maintain a consistent sign across all time steps. For `AlignPost` projections, $\mathbf{x}$ denotes the binary spike vector (see Eqs. 36-37); while for `AlignPre` projections, $\mathbf{x}^t$ denotes the synaptic conductance (see Eqs. 34-35). In both cases, the elements in $\mathbf{x}^t$ are sign-preserving and consistently positive across all time steps. Additionally, for the monotonous activation function f, its derivative $\mathbf{D}_f^t$ also maintains a consistent sign throughout all times. This distinctive property leads to an important mathematical implication: the summation of multiple rank-one matrices can be approximated by the rank-one outer product of the summed vectors. Theorem 7.3 formalizes this concept, with its theoretical proof presented in Methods 7.5 and empirical validation on various SNN dynamics using real-world neuromorphic datasets provided in Methods 7.6.

Theorem 7.3. *Let $\mathbf{a}_i \in \mathbb{R}^m$ and $\mathbf{b}_i \in \mathbb{R}^n$ be vectors where all elements in $\mathbf{a}_i$ and $\mathbf{b}_i$ are of the same sign (i.e., they are all positive or negative). Then, the cosine similarity between $\sum_{i=1}^r \mathbf{a}_i \otimes \mathbf{b}_i$ and $\left(\frac{1}{r}\sum_{i=1}^r \mathbf{a}_i\right) \otimes \sum_{i=1}^r \mathbf{b}_i$ approaches 1 as $r \to \infty$:*

$$\lim_{r\to\infty} \cos\left(\sum_{i=1}^r \mathbf{a}_i \otimes \mathbf{b}_i, \left(\frac{1}{r}\sum_{i=1}^r \mathbf{a}_i\right) \otimes \sum_{i=1}^r \mathbf{b}_i\right) \to 1. \tag{55}$$

7.4 Proof of Theorem 7.2

Proof. Given Eq. 1, using the chain rule, we can write the gradient as:

[2] $\mathbf{D}_f^t$ is actually a block diagonal matrix of dimensions $\mathbb{R}^{Hd\times H}$, consisting of H-by-H blocks. Each diagonal block, $\mathbf{D}_{f,ii}^t$, is a d-by-1 matrix. For examples, refer to Methods 7.2. With loss of generality, all subsequent derivations assume $d = 1$, making $\mathbf{D}_f^t$ a diagonal matrix of dimensions $\mathbb{R}^{H\times H}$.

$$\frac{\partial \mathbf{h}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathbf{h}}{\partial \hat{\mathbf{h}}} \frac{\partial \hat{\mathbf{h}}}{\partial \boldsymbol{\theta}}$$

where:

- $\mathbf{h} = f(\hat{\mathbf{h}}) \in \mathbb{R}^H$ by definition of the mapping
- $\mathbf{I} = \boldsymbol{\theta} \circledast \mathbf{x} \in \mathbb{R}^H$ is the pre-activation

Calculating each component:

- $\frac{\partial \mathbf{h}_i}{\partial \mathbf{I}_i} = f'(\mathbf{I}_i)$, $\frac{\partial \mathbf{h}_i}{\partial \mathbf{I}_j} = 0$ $(i \neq j)$, since f is an element-wise activation function. Therefore, $\frac{\partial \mathbf{h}}{\partial \mathbf{I}} = \mathbf{D}_f \in \mathbb{R}^{H \times H}$, where $\mathbf{D}_f$ is a diagonal matrix, and $D_{f,ii} = f'(\mathbf{I}_i)$ by the derivative of f.
- $\frac{\partial \mathbf{I}_j}{\partial \boldsymbol{\theta}_{ij}} = \mathbf{x}_i$. In the vector form, $\frac{\partial \mathbf{I}}{\partial \boldsymbol{\theta}_{ij}} = \mathbf{x}$ by the definition of matrix multiplication.

Substituting back into the chain rule:

$$\frac{\partial \mathbf{h}_j}{\partial \boldsymbol{\theta}_{ij}} = f'(\mathbf{I}_j) \frac{\partial \mathbf{I}_j}{\partial \boldsymbol{\theta}_{ij}} = f'(\mathbf{I}_j)\mathbf{x}_i$$

In the vector form, we got

$$\frac{\partial \mathbf{h}}{\partial \boldsymbol{\theta}} = \mathbf{D}_f \otimes \mathbf{x}$$

where $\otimes$ is the outer product.

□

7.5 Proof of Theorem 7.3

Theorem 7.4. *Let $\mathbf{a}_i \in \mathbb{R}^m$ and $\mathbf{b}_i \in \mathbb{R}^n$ be vectors. Elements in $\mathbf{a}_i$ are sampled from the distribution with mean $\mathbf{a}$ and variance $\sigma_{\mathbf{a}}^2$, i.e., $\mathbf{a}_i = \mathbf{a} + \sigma_{\mathbf{a}}\boldsymbol{\xi}_i$, $\langle \boldsymbol{\xi}_i \rangle = \mathbf{0}$, $\langle \boldsymbol{\xi}_i^T \boldsymbol{\xi}_j \rangle = m\delta_{i,j}$. Elements in $\mathbf{b}_i$ are sampled from the distribution with mean $\mathbf{b}$ and variance $\sigma_{\mathbf{b}}^2$, i.e., $\mathbf{b}_i = \mathbf{b} + \sigma_{\mathbf{b}}\boldsymbol{\xi}_i$, $\langle \boldsymbol{\xi}_i \rangle = \mathbf{0}$, $\langle \boldsymbol{\xi}_i^T \boldsymbol{\xi}_j \rangle = n\delta_{i,j}$. Then, as $r \to \infty$, the cosine similarity between $\frac{1}{r}\sum_{i=1}^r \mathbf{a}_i \otimes \mathbf{b}_i$ and $\left(\frac{1}{r}\sum_{i=1}^r \mathbf{a}_i\right) \otimes \left(\frac{1}{r}\sum_{i=1}^r \mathbf{b}_i\right)$ has the form of:*

$$\begin{aligned} &\lim_{r\to\infty} \cos\left(\frac{1}{r}\sum_{i=1}^r \mathbf{a}_i \otimes \mathbf{b}_i, \left(\frac{1}{r}\sum_{i=1}^r \mathbf{a}_i\right) \otimes \left(\frac{1}{r}\sum_{i=1}^r \mathbf{b}_i\right)\right) \\ &\approx \sqrt{1 - \frac{1}{r r_{\mathbf{a}} r_{\mathbf{b}} + r_{\mathbf{a}} + r_{\mathbf{b}} + 1}}, \\ &\to 1, \end{aligned} \tag{56}$$

when $r_{\mathbf{a}} = \frac{\mathbf{a}^T\mathbf{a}}{m\sigma_{\mathbf{a}}^2} \neq 0$ and $r_{\mathbf{b}} = \frac{\mathbf{b}^T\mathbf{b}}{n\sigma_{\mathbf{b}}^2} \neq 0$.

Proof. Let $\mathbf{A} = \frac{1}{r}\sum_{i=1}^r \mathbf{a}_i$, and $\mathbf{B} = \frac{1}{r}\sum_{i=1}^r \mathbf{b}_i$.

Then, we have:

$$\begin{aligned} \mathbb{E}\left[\mathbf{a}_i^T\mathbf{A}\right] &= \mathbb{E}\left[\mathbf{A}^T\mathbf{A}\right] = \mathbf{a}^T\mathbf{a} + \frac{m\sigma_{\mathbf{a}}^2}{r}, \\ \mathbb{E}\left[\mathbf{b}_i^T\mathbf{B}\right] &= \mathbb{E}\left[\mathbf{B}^T\mathbf{B}\right] = \mathbf{b}^T\mathbf{b} + \frac{n\sigma_{\mathbf{b}}^2}{r} \\ \mathbb{E}\left[\sum_{i,j} \mathbf{a}_i^T\mathbf{a}_j\mathbf{b}_i^T\mathbf{b}_j\right] &= (r^2 - r)(\mathbf{a}^T\mathbf{a})(\mathbf{b}^T\mathbf{b}) + r(\mathbf{a}^T\mathbf{a} + m\sigma_{\mathbf{a}}^2)(\mathbf{b}^T\mathbf{b} + n\sigma_{\mathbf{b}}^2). \end{aligned}$$

The cosine similarity between two vectors $\mathbf{u}$ and $\mathbf{v}$ is defined as:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^T \mathbf{v}}{|\mathbf{u}||\mathbf{v}|},$$

where $|\cdot|$ denotes the L2 norm.

Then, consider the cosine similarity between $\frac{1}{r}\sum_{i=1}^{r} \mathbf{a}_i \otimes \mathbf{b}_i$ and $\mathbf{A} \otimes \mathbf{B}$:

$$\begin{aligned}
\cos\left(\frac{1}{r}\sum_{i=1}^{r} \mathbf{a}_i \otimes \mathbf{b}_i, \mathbf{A} \otimes \mathbf{B}\right) &= \frac{\frac{1}{r}\sum_i^r (\mathbf{a}_i^T \mathbf{A})(\mathbf{b}_i^T \mathbf{B})}{|\frac{1}{r}\sum_i^r \mathbf{a}_i \otimes \mathbf{b}_i||\mathbf{A} \otimes \mathbf{B}|}, \\
&= \frac{\sum_i^r (\mathbf{a}_i^T \mathbf{A})(\mathbf{b}_i^T \mathbf{B})}{\sqrt{\left(\sum_{i,j} (\mathbf{a}_i^T \mathbf{a}_j)(\mathbf{b}_i^T \mathbf{b}_j)\right)(\mathbf{A}^T \mathbf{A})(\mathbf{B}^T \mathbf{B})}}, \\
&\approx \frac{r(\mathbf{a}^T \mathbf{a} + \frac{m\sigma_{\mathbf{a}}^2}{r})(\mathbf{b}^T \mathbf{b} + \frac{n\sigma_{\mathbf{b}}^2}{r})}{\sqrt{r^2(\mathbf{a}^T \mathbf{a})(\mathbf{b}^T \mathbf{b}) + r(n\sigma_{\mathbf{b}}^2 \mathbf{a}^T \mathbf{a} + m\sigma_{\mathbf{a}}^2 \mathbf{b}^T \mathbf{b} + mn\sigma_{\mathbf{a}}^2 \sigma_{\mathbf{b}}^2)}}, \\
&= \sqrt{\frac{(\mathbf{a}^T \mathbf{a} + \frac{m\sigma_{\mathbf{a}}^2}{r})(\mathbf{b}^T \mathbf{b} + \frac{n\sigma_{\mathbf{b}}^2}{r})}{(\mathbf{a}^T \mathbf{a})(\mathbf{b}^T \mathbf{b}) + \frac{1}{r}(n\sigma_{\mathbf{b}}^2 \mathbf{a}^T \mathbf{a} + m\sigma_{\mathbf{a}}^2 \mathbf{b}^T \mathbf{b} + mn\sigma_{\mathbf{a}}^2 \sigma_{\mathbf{b}}^2)}}, \\
&= \sqrt{1 + \frac{(\frac{1}{r^2} - \frac{1}{r})mn\sigma_{\mathbf{a}}^2 \sigma_{\mathbf{b}}^2}{(\mathbf{a}^T \mathbf{a})(\mathbf{b}^T \mathbf{b}) + \frac{1}{r}(n\sigma_{\mathbf{b}}^2 \mathbf{a}^T \mathbf{a} + m\sigma_{\mathbf{a}}^2 \mathbf{b}^T \mathbf{b} + mn\sigma_{\mathbf{a}}^2 \sigma_{\mathbf{b}}^2)}}, \\
&= \sqrt{1 + \frac{(\frac{1}{r^2} - \frac{1}{r})}{r_{\mathbf{a}} r_{\mathbf{b}} + \frac{1}{r}(r_{\mathbf{a}} + r_{\mathbf{b}} + 1)}}.
\end{aligned} \tag{57}$$

When $r \to \infty$, $\frac{1}{r^2} \to 0$, and Eq. 57 can be approximated by:

$$\cos\left(\frac{1}{r}\sum_{i=1}^{r} \mathbf{a}_i \otimes \mathbf{b}_i, \mathbf{A} \otimes \mathbf{B}\right) \approx \sqrt{1 - \frac{1}{r r_{\mathbf{a}} r_{\mathbf{b}} + r_{\mathbf{a}} + r_{\mathbf{b}} + 1}}. \tag{58}$$

When $r_{\mathbf{a}} \neq 0$ and $r_{\mathbf{b}} \neq 0$,

$$\lim_{r \to \infty} \cos\left(\frac{1}{r}\sum_{i=1}^{r} \mathbf{a}_i \otimes \mathbf{b}_i, \mathbf{A} \otimes \mathbf{B}\right) = 1. \tag{59}$$

□

Given the above Theorem 7.4, now let us give a formal proof of Theorem 7.3.

Proof of Theorem 7.3. Since $\mathbf{a}_i \in \mathbb{R}^m$ and $\mathbf{b}_i \in \mathbb{R}^n$ be vectors where all elements in $\mathbf{a}_i$ and $\mathbf{b}_i$ are of the same sign (i.e., they are all positive or negative), we have:

$$\mathbb{E}[\mathbf{a}_i] \neq 0,$$
$$\mathbb{E}[\mathbf{b}_i] \neq 0.$$

Then we have:

$$\frac{\mathbf{a}^T \mathbf{a}}{m\sigma_{\mathbf{a}}^2} \neq 0,$$
$$\frac{\mathbf{b}^T \mathbf{b}}{n\sigma_{\mathbf{b}}^2} \neq 0.$$

According to Theorem 7.4, we have:

$$\lim_{r\to\infty} \cos\left(\sum_{i=1}^{r} \mathbf{a}_i \otimes \mathbf{b}_i, \left(\frac{1}{r}\sum_{i=1}^{r} \mathbf{a}_i\right) \otimes \sum_{i=1}^{r} \mathbf{b}_i\right) \to 1.$$

□

The cosine similarity is close to 1 means that the resulting matrices are pointing in almost the same direction. In the context of Jacobian matrices, this implies that the rate of change of one set of variables with respect to another set of variables is highly similar between the two matrices.

7.6 Experimental validations of Theorem 7.3

To verify the theoretical analysis in above, we conducted two types of numerical experiments.

First, we performed experiments using randomly generated datasets that mimic the conditions a real synaptic projection might process. Specifically, we assumed that presynaptic neurons generated spikes according to a binomial distribution with probability p, and the derivative of the activation function f' followed a normal distribution $\mathcal{N}(\mu_f, \sigma_f^2)$. We fixed the $\mu_f = 1.0$, $m = 10$, and $n = 10$. Then, we systematically varied σ_f and p, generated corresponding r random samples, and computed the cosine similarity between $\sum_{i=1}^{r} \mathbf{x}_i \otimes \mathbf{D}_f^i$ and $\left(\frac{1}{r}\sum_{i=1}^{r} \mathbf{x}_i\right) \otimes \sum_{i=1}^{r} \mathbf{D}_f^i$ (see Theorem 7.3) using the generated samples. For each parameter pair, we used $r = 10,000$. The resulting phase portrait is visualized in Figure S9A. We also computed the theoretical cosine similarity using Eq. 58. These theoretical results were plotted in Figure S9B. As we can see, the theoretical and experimental results matched very well.

Next, we calculated the cosine similarity between $\sum_{i=1}^{r} \mathbf{x}_i \otimes \mathbf{D}_f^i$ and $\left(\frac{1}{r}\sum_{i=1}^{r} \mathbf{x}_i\right) \otimes \sum_{i=1}^{r} \mathbf{D}_f^i$ using diverse SNN dynamics to process the IBM DVS Gestures dataset [3]. We examined how the cosine similarity changed as the state evolved forward over time. For each experiment, we computed 500 data samples, and reported the mean and 95% confidence interval of cosine similarities across these samples. Experimental results are plotted in Figure S9C-J. We observed that, over the entire sequence, the rank-one approximation of most eligibility traces maintained a very high cosine similarity (>80%). Moreover, there were two types of eligibility traces. In the first type, the cosine similarity gradually decayed over time. They correspond to the eligibility traces of fast decaying variables, such as the synaptic conductance $\mathbf{g}$ and the membrane potential $\mathbf{v}$ in models presented in Methods 7.2 (please also refer to the lines of "$\mathbf{x} \otimes f_{\mathbf{g}}$" and "$\mathbf{x} \otimes f_{\mathbf{v}}$" in Figure S9C-J). In the second type, the cosine similarity started very low, then rapidly increased to approach 1.0 as time progressed, and remained high as the network states evolved. These eligibility traces track the neuronal and synaptic variables with slow times constants, such as the slow adaption current $\mathbf{a}$ and the short-term depression variable $\mathbf{x}$ in models presented in Methods 7.2 (please also refer to the lines of "$\mathbf{x} \otimes f_{\mathbf{x}}$" and "$\mathbf{x} \otimes f_{\mathbf{a}}$" in Figure S9E-J). Overall, our empirical experiment demonstrated that the rank-one approximation can effectively capture the essential features of eligibility traces of most variables over long time sequences, and even achieve a perfect reproduction of eligibility traces for those variables with slow dynamics.

7.7 Derivation of Online Learning Algorithms

In this section, we provide the details of the algorithmic derivations underpinning our online learning algorithms.

7.7.1 Theoretical basics of RTRL

Let us consider biological SNN models whose dynamics are governed by Eq. 1. At each time step $t \in \mathcal{T}$, we assume that the state is transformed into an output $\mathbf{y}^t = f_o(\mathbf{h}^t, \boldsymbol{\phi})$, while the network receives a loss $\mathcal{L}^t(\mathbf{y}^t, \hat{\mathbf{y}}^t)$. The objective is to optimize the total loss $\mathcal{L} = \sum_t \mathcal{L}^t$ with respect to the parameters $\boldsymbol{\theta}$, utilizing the gradient

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}} = \sum_{t\in\mathcal{T}} \sum_{k=1}^{t} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^k} \frac{\partial \mathbf{h}^k}{\partial \boldsymbol{\theta}^k}, \tag{60}$$

where $\boldsymbol{\theta}^k$ represents the parameters utilized at time t, $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^k}\frac{\partial \mathbf{h}^k}{\partial \boldsymbol{\theta}^k}$ measures how $\boldsymbol{\theta}$ at step k affects the loss at step $t \geq k$, and the factor $\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^k}$ transports the error through time from step t to step k. When $k \ll t$, the transport captures a long-term dependency, otherwise a short-term dependency.

To compute Eq. 60, two methods are commonly used, that is BPTT [78, 41, 79] and RTRL [58, 80, 81], which iteratively perform the error propagation on the unfolded computation graph through time backwardly and forwardly, respectively (also refer to Figure S7).

For the BPTT, the gradient is solved iteratively, spanning from the final time step to the first one:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}}{\partial \mathbf{h}^t}\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t} = \sum_{t\in\mathcal{T}} \Phi^t \frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t}, \tag{61}$$

$$\Phi^t = \Phi^{t+1}\frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t} + \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}, \tag{62}$$

where $\Phi \in \mathbb{R}^H$ defines the partial gradients of the loss with respect to the hidden state. Here, we only use the batch size $B = 1$.

For the RTRL, the gradient is updated in the forward direction:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}\boldsymbol{\epsilon}^t, \tag{63}$$

$$\boldsymbol{\epsilon}^t = \frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}}\boldsymbol{\epsilon}^{t-1} + \frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t}, \tag{64}$$

where the influence matrix $\boldsymbol{\epsilon} \in \mathbb{R}^{H\times H\times I}$ calculates the effect of the hidden units on the parameters.

To better illustrate the difference between BPTT and RTRL, let's examine the gradients of the loss at time t with respect to the parameter at time k, denoted as $\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}^k}$, where $t \geq k$. Figure S7 demonstrates that BPTT's backward computation of gradients is significantly more efficient than RTRL's forward computation. This is because the product of Jacobians from the end to the beginning of the composed function only requires $\mathcal{O}(H)$ memory storage. In contrast, RTRL's memory complexity is $\mathcal{O}(H \times H \times I)$ as it computes Jacobian products forwardly. To reduce the intrinsic complexity of RTRL, we should take full consideration of SNN properties.

7.7.2 Derivation of the D-RTRL algorithm from RTRL

In the following sections, we will concentrate on the scenario where the hidden state dimension $d = 1$ to develop the D-RTRL algorithm. This simplification enables us to clearly demonstrate the core principles of the algorithm without the additional complexity introduced by multi-dimensional hidden states. For learning rules applicable when ($d > 1$), please refer to Methods 7.7.4.

As stated in the above sectons, the raw RTRL has a high memory complexity. By substituting Eq. 53 and Eq. 54 into Eq. 64, we can approximate $\boldsymbol{\epsilon}^t$ as:

$$\boldsymbol{\epsilon}^t \approx \underbrace{\mathbf{D}^t}_{\approx \partial \mathbf{h}^t/\partial \mathbf{h}^{t-1}} \boldsymbol{\epsilon}^{t-1} + \underbrace{\mathbf{D}_f^t \otimes \mathbf{x}^t}_{=\partial \mathbf{h}^t/\partial \boldsymbol{\theta}^t}. \tag{65}$$

Eq. 65 results in a very sparse matrix with $\mathcal{O}(H \times H \times I)$ memory complexity. For compact storage, the above equation is equivalent to the following form:

$$\boldsymbol{\epsilon}^t \approx \mathbf{D}^t\boldsymbol{\epsilon}^{t-1} + \mathrm{diag}(\mathbf{D}_f^t) \otimes \mathbf{x}^t, \tag{66}$$

where $\mathrm{diag}(\mathbf{D}_f^t) \in \mathbb{R}^H$ is the vector storing the diagonal elements of $\mathbf{D}_f^t$. This results in $\boldsymbol{\epsilon}^t$ having the shape of $\mathbb{R}^{H\times I}$.

In practice, $\mathbf{D}^t$ is stored as a vector $\mathrm{diag}(\mathbf{D}^t) \in \mathbb{R}^{H\times 1}$. Consequently, Eq. 66 can be reformulated using element-wise multiplication. Combining with Eq. 63, we can express the learning rule of the D-RTRL algorithm as:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}}\mathcal{L} &\approx \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \circ \boldsymbol{\epsilon}^t, \\ \boldsymbol{\epsilon}^t &\approx \mathrm{diag}(\mathbf{D}^t) \circ \boldsymbol{\epsilon}^{t-1} + \mathrm{diag}(\mathbf{D}_f^t) \otimes \mathbf{x}^t. \end{aligned} \tag{67}$$

where $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \in \mathbb{R}^{H\times 1}$ is a column vector.

D-RTRL generally exhibits a memory complexity of $\mathcal{O}(BP)$, where B represents the batch size and P denotes the number of parameters. This efficient scaling is primarily due to the nature of the eligibility trace $\boldsymbol{\epsilon}^t$, which only needs to store the historical information for each active synaptic connection. The memory efficiency of D-RTRL becomes particularly advantageous when dealing with sparse neural networks, a common characteristic in biologically inspired models. In sparse networks, many synaptic connections have zero weights. D-RTRL capitalizes on this sparsity by eliminating the need to store eligibility traces for these zero-weight connections. This optimization can lead to significant memory savings, especially in large-scale networks where sparsity is pronounced. In convolutional layers, D-RTRL exhibits greater memory efficiency since the parameter size is typically smaller than the number of presynaptic and postsynaptic neurons for small-sized convolutional kernels.

Moreover, the $\mathcal{O}(BP)$ complexity of D-RTRL represents a substantial improvement over the $\mathcal{O}(BH^2)$ complexity of previous online algorithms such as E-prop [6] and OSTL [8], where H is the number of hidden units. This reduction in complexity enables D-RTRL to scale more effectively to larger networks and longer temporal sequences, making it a promising approach for training complex spiking networks.

7.7.3 Derivation of the ES-D-RTRL algorithm from D-RTRL

In this section, we derive the learning rule for the ES-D-RTRL algorithm by focusing on the scenario where the hidden state dimension $d = 1$. The derived learning rule can naturally extend to cases where $d > 1$. For more details on the generalization to multi-dimensional hidden states, please refer to Methods 7.7.4.

Given the learning rule of the D-RTRL algorithm in Eq. 67, let's consider the influence of $\boldsymbol{\theta}$ at step k to the loss $\mathcal{L}$ at step t $(t \geq k)$. It can be approximated by:

$$\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}^k} \approx \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \underbrace{\left(\prod_{i=k+1}^{t} \mathbf{D}^i\right)}_{\approx \partial \mathbf{h}^t / \partial \mathbf{h}^k} \underbrace{\left(\mathbf{D}_f^k \otimes \mathbf{x}^k\right)}_{= \partial \mathbf{h}^k / \partial \boldsymbol{\theta}^k}. \tag{68}$$

Therefore, at the time t, the parameter gradient accumulated up to time t has the form of:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}^t = \sum_{k=1}^{t} \frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}^k} \approx \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \sum_{k=1}^{t} \left(\prod_{i=k+1}^{t} \mathbf{D}^i\right) \left(\mathbf{D}_f^k \otimes \mathbf{x}^k\right). \tag{69}$$

Here, we assume that a learning target is provided at each time step $k \in [1, \cdots, t]$. Moreover, $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \in \mathbb{R}^{H}$ is defined as a row vector in Eqs. 68 and 69.

Since $\mathbf{D}^i$ and $\mathbf{D}_f$ are diagonal matrices, we can only keep their diagonal elements as vectors, and their matrix multiplication operations are equivalent to the element-wise Hadamard products of diagonal vectors:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}^t \approx \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \circ \left[\sum_{k=1}^{t} \left(\underbrace{\mathrm{diag}(\mathbf{D}^t) \circ \cdots \circ \mathrm{diag}(\mathbf{D}^{k+1}) \circ \mathrm{diag}(\mathbf{D}_f^k)}_{\text{postsynaptic activity}}\right) \otimes \underbrace{\mathbf{x}^k}_{\text{presynaptic activity}}\right]. \tag{70}$$

From this equation onward, $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \in \mathbb{R}^{H\times 1}$ is defined as a column vector.

Eq. 70 results in significant memory overhead because the summation requires storing t pairs of vectors representing pre- and postsynaptic activity. This memory requirement is equivalent to BPTT, with complexity scaling linearly as time progresses.

We found a solution to this issue by fully leveraging the unique properties of SNNs. The key insight lies in the nature of neuronal output of SNNs: (1) Binary events: neurons in SNNs output binary signals (spikes); (2) Non-negativity: these spike events are always non-negative. These properties are detailed in Methods 7.3, and the foundation of our solution is provided by Theorem 7.3. This theorem allows us to approximate Eq. 70 by

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}^t \approx \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \circ \left(\frac{1}{t} \sum_{k=1}^{t} \mathbf{b}_k^t \otimes \sum_{k=1}^{t} \mathbf{x}^k\right), \tag{71}$$

where $\mathbf{b}_k^t = \mathrm{diag}(\mathbf{D}^t) \circ \cdots \circ \mathrm{diag}(\mathbf{D}^{k+1}) \circ \mathrm{diag}(\mathbf{D}_f^k)$.

To avoid of the exploding problem, the eligibility trace should prioritize recent activities rather than retain a complete historical record over a very long sequence. We implemented this by modifying Eq. 71 using two key techniques: exponential smoothing for $\mathbf{b}^t$ and a low-pass filter for $\mathbf{x}$. The former computes a running average of $\mathbf{b}^t$ online, while the latter calculates a running summation of $\mathbf{x}$. These adaptations form the foundation of our ES-D-RTRL algorithm, whose learning rule can be expressed as follows:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathcal{L} &\approx \sum_{t=\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \circ \left(\boldsymbol{\epsilon}_{\mathbf{f}}^t \otimes \boldsymbol{\epsilon}_{\mathbf{x}}^t \right), \\ \boldsymbol{\epsilon}_{\mathbf{x}}^t &= \alpha \boldsymbol{\epsilon}_{\mathbf{x}}^{t-1} + \mathbf{x}^t, \\ \boldsymbol{\epsilon}_{\mathbf{f}}^t &= \alpha\, \mathrm{diag}(\mathbf{D}^t) \circ \boldsymbol{\epsilon}_{\mathbf{f}}^{t-1} + (1-\alpha) \mathrm{diag}(\mathbf{D}_f^t), \end{aligned} \tag{72}$$

where α $(0 < \alpha < 1)$ is the smoothing factor, corresponding to the time constant $\tau = -\Delta t / \ln(\alpha)$. The only

7.7.4 Generalization to hidden states with multiple variables ($d > 1$)

It is important to note that the derivations presented in Methods 7.7.2 and 7.7.3 are based on the simplifying assumption that $d = 1$, where d represents the number of variables in each neuron's hidden states. However, these derivations can be generalized to cases where $d > 1$ without loss of generality.

When $d > 1$, the following adjustments to the matrix dimensions and structures are necessary:

- $\mathbf{h}^t$: The hidden state becomes a block matrix with dimensions $\mathbb{R}^{Hd \times 1}$, where H is the number of hidden units. It contains H blocks with each block is a matrix of shape $\mathbb{R}^{d \times 1}$. The block $\mathbf{h}_i$ belongs to the state variables of neuron i (see examples in Methods 7.2).
- $\mathbf{D}^t$: This matrix becomes a block diagonal matrix with dimensions $\mathbb{R}^{Hd \times Hd}$. It consists of H blocks along the diagonal, each block being a $d \times d$ matrix. Each diagonal block $\mathbf{D}_{ii}^t$ belongs to the Jacobian matrix between d state variables of neuron i (see examples in Methods 7.2).
- $\mathbf{D}_f^t$: This matrix transforms into a block diagonal matrix of shape $\mathbb{R}^{Hd \times H}$. It comprises H blocks along the diagonal, where each block is a matrix of shape $\mathbb{R}^{d \times 1}$. Each diagonal block $\mathbf{D}_{f,ii} \in \mathbb{R}^{b \times 1}$ belongs to the hidden-to-weight Jacobian $\partial \mathbf{h}_i^t / \partial \mathbf{I}_i^t$ of neuron i (see examples in Methods 7.2).
- $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}$: The gradient of the loss with respect to the hidden state becomes a block matrix of shape $\mathbb{R}^{Hd \times 1}$. It consists of H blocks, each with dimensions $\mathbb{R}^{d \times 1}$.

As we can see, in the case of multi-dimensional hidden states ($d > 1$), the matrices in Eqs. 67 and 72 transform into block matrices. Each element of these block matrices is no longer a scalar, but rather a sub-matrix or vector. However, the transition from scalar ($d = 1$) to multi-dimensional ($d > 1$) hidden states maintains the overall online learning framework as shown in Eqs. 67 and 72. The only difference is on the $\circ$ operations in the aforementioned equations.

For $d = 1$, $\circ$ represents the ***element-wise scalar multiplication***. This is because $\mathbf{h}_i^t$, $\mathbf{D}_{ii}^t$, $\mathbf{D}_{f,ii}^t$, and $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}_i^t}$ are all scalar values.

For $d > 1$, $\circ$ evolves into a ***block-wise multiplication*** operation. In this context, the multiplication is performed on corresponding blocks, with each block undergoing d-dimensional matrix or inner multiplication.

To elucidate this distinction, let us examine the learning rule for a single weight $\boldsymbol{\theta}_{ji}$ connecting neurons i and j. For the D-RTRL algorithm, the block-wise implementation of Eq. 67 becomes:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}_{ji}} \mathcal{L} &\approx \sum_{t \in \mathcal{T}} \left\langle \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}_j^t}\ ,\ \boldsymbol{\epsilon}_{ji}^t \right\rangle, \\ \boldsymbol{\epsilon}_{ji}^t &\approx \mathbf{D}_{jj}^t \boldsymbol{\epsilon}_{ji}^{t-1} + \mathbf{D}_{f,jj}^t \cdot {\mathbf{x}_i}^t, \end{aligned} \tag{73}$$

where ${\mathbf{x}_i}^t \in \mathbb{R}$ is a scalar value, $\boldsymbol{\epsilon}^t \in \mathbb{R}^{Hd\times H}$ now becomes a block matrix with H-by-H blocks and each block $\boldsymbol{\epsilon}^t_{ji}$ is a vector with the shape of $\mathbb{R}^{d\times 1}$, and $\langle \cdot\ ,\ \cdot \rangle : \mathbb{R}^{d\times 1} \times \mathbb{R}^{d\times 1} \to \mathbb{R}$ is the inner product between two vectors.

Similarly, for the ES-D-RTRL algorithm, the learning rule in Eq. 72 is implemented block-wise as:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}_{ji}}\mathcal{L} &\approx \sum_{t=\mathcal{T}} \left\langle \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t_j}\ ,\ \boldsymbol{\epsilon}^t_{\mathbf{f},j} \right\rangle \cdot \boldsymbol{\epsilon}^t_{\mathbf{x},i}, \\ \boldsymbol{\epsilon}^t_{\mathbf{f},j} &= \alpha\, \mathbf{D}^t_{jj}\boldsymbol{\epsilon}^{t-1}_{\mathbf{f},j} + (1-\alpha)\mathbf{D}^t_{f,jj}, \end{aligned} \tag{74}$$

where $\boldsymbol{\epsilon}^t_{\mathbf{f}} \in \mathbb{R}^{Hd\times 1}$ now becomes a block matrix with each block $\boldsymbol{\epsilon}^t_{\mathbf{f},j}$ a vector with the shape of $\mathbb{R}^{d\times 1}$, $\boldsymbol{\epsilon}^t_{\mathbf{x},i} \in \mathbb{R}$ is a scalar value that tracking the eligibility trace of presynaptic neuron i.

These formulations highlight the critical differences in computational operations between scalar and multi-dimensional hidden states. The transition from element-wise to block-wise operations not only affects the mathematical representation but also has implications for computational efficiency and implementation strategies in practice. Furthermore, this generalization to multi-dimensional hidden states opens up new possibilities for capturing more complex temporal dependencies and representations in SNNs, potentially leading to enhanced performance in various sequence modeling tasks.

7.7.5 Learning signals in deep recurrent layers

In a network with L stacked recurrent layers, where $\mathbf{h}^t = [\mathbf{h}^t_1, \cdots, \mathbf{h}^t_L]$ represents the hidden states at time t, the computation of exact eligibility traces is inherently very complex [81]. This complexity arises from the interdependencies between layers and the propagation of gradients through time. To maintain the simplicity and efficiency of the algorithms discussed in Section 3, we introduce an approximation that simplifies these calculations.

Our approach involves ignoring the influence of hidden states in deeper layers ($\mathbf{h}_{l+1}, \cdots, \mathbf{h}_L$) on the parameters of the l-th layer, denoted as $\boldsymbol{\theta}_l$. Instead, we focus on maintaining the eligibility trace $\boldsymbol{\epsilon}^t_l$ with respect to its own hidden state $\mathbf{h}_l$ (Figure 3D). This simplification allows for more tractable computations while still capturing the essential dynamics of the recurrent network.

When a top-down learning signal reaches layer l at time $t \in \mathcal{T}$, we can compute its parameter gradients using the following equation:

$$\nabla_{\boldsymbol{\theta}_l}\mathcal{L} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t_l} \circ \boldsymbol{\epsilon}^t_l. \tag{75}$$

7.7.6 Theoretical analysis of memory efficiency and computational speed

The summary of memory and time complexity of the evaluated algorithms is presented in Figure 5I.

BPTT has a memory complexity of $\mathcal{O}(TBN)$, where T is the number of time steps unrolled during training, B is the batch size, and N is the hidden state size. Generally, the memory complexity of RTRL is cubic in the number of hidden units at a single time step, making it impractical for large networks. However, when $T > N^2$, RTRL shows a memory advantage over BPTT. E-prop [6], an approximation of RTRL, provides a biologically plausible learning rule for spiking networks but has a memory complexity of $\mathcal{O}(BN^2)$ for maintaining the eligibility trace. OSTL [8] shares the same memory complexity as E-prop. Both E-prop and OSTL are designed for densely connected recurrent layers. In contrast, our D-RTRL has a memory complexity proportional to the parameter size P, achieving $\mathcal{O}(BP)$ complexity. For densely connected layers, D-RTRL matches the memory complexity of E-prop and OSTL at $\mathcal{O}(BN^2)$. For sparsely connected layers, D-RTRL is more memory-efficient than E-prop and OSTL, as it only stores the eligibility trace at non-zero weight positions. In convolutional layers, D-RTRL exhibits greater memory efficiency since the parameter size is typically smaller than the number of presynaptic and postsynaptic neurons for small-sized convolutional kernels. ES-D-RTRL shows a memory complexity of $\mathcal{O}(BN)$, scaling linearly with the number of neurons in the network because it only needs to store two traces of presynaptic and postsynaptic activities (refer to Eq. 5-6). For densely connected layers, ES-D-RTRL is more

memory-efficient than D-RTRL, as the number of pre-synaptic and post-synaptic neurons is usually smaller than the number of connections between them. However, for convolutional layers with small kernels, D-RTRL may be more memory-efficient than ES-D-RTRL.

The computational complexity of BPTT is relatively low because the backward gradient computation at each time step involves a batched VJP with a complexity of $\mathcal{O}(BN^2)$. Raw RTRL has a time complexity of $\mathcal{O}(BN^4)$ per step, making it nearly impossible to use for training networks with $N > 1000$. Both OSTL and E-prop have a time complexity of $\mathcal{O}(BN^2)$. The computational complexity of D-RTRL depends on the size of $\boldsymbol{\theta}$. For densely connected layers, it is $\mathcal{O}(BN^2)$; for sparsely connected layers, it is $\mathcal{O}(pBN^2)$, where p is the connection probability; for convolutional layers, the time complexity depends on the size of the convolutional kernels. It is important to note that while BPTT and D-RTRL both have a theoretical complexity of $\mathcal{O}(BN^2)$ for dense connection layers, they differ in practice. In BPTT, it is a matrix-matrix multiplication, a computation-bound operation, while in D-RTRL, it is a batched matrix-vector element-wise multiplication (refer to Figure 3B and Eq. 3), a memory-bound operation. The ES-D-RTRL algorithm has a computational complexity of $\mathcal{O}(BN)$, which are element-wise operations among presynaptic and postsynaptic neuronal activities (refer to Figure 3C and Eq. 6).

7.8 Experimental Details

This section provides an overview of the experimental methodology employed in our study.

7.8.1 Surrogate gradient function

Translating membrane potential fluctuations into discrete spike events, as described by Eq. 9, relies on a Heaviside step function with its transition point at the neuron's firing threshold v_{th}. However, this function presents challenges for gradient-based learning algorithms due to its discontinuity at the threshold and lack of meaningful gradients elsewhere. To overcome these limitations and enable effective training of spiking networks, we employed a surrogate gradient approach used in previous works [84].

During the forward pass, we use the Heaviside function to generate the spike:

$$\mathbf{z}^t = \text{spike}(x) = \mathcal{H}(\mathbf{v}^t - v_{\text{th}}),$$

where x is used to represent $\mathbf{v}^t - v_{\text{th}}$.

During the backward pass, we approximated the derivative of the Heaviside step function with the ReLU function:

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

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. The derivative of the spike function, $\text{spike}'(x)$, is a triangular function centered at $x = 0$. It reaches its maximum value of $\alpha * \text{width}$ at $x = 0$ and decreases linearly to zero at $x = \pm\text{width}$. When $|x| > \text{width}$, $\text{spike}'(x) = 0$.

7.8.2 The leaky rate readout

All network architectures incorporate a specialized leaky readout mechanism, consisting of dedicated output neurons corresponding to each task-specific label or behavior. These output neurons receive linear projections from the recurrent layer. The temporal evolution of the readout neurons is governed by the following differential equation:

$$\tau_{\text{out}} \frac{d\mathbf{y}}{dt} = -\mathbf{y} + W^{\text{out}} \mathbf{r}^{\text{rec}} + b^{\text{out}}. \tag{76}$$

In this equation, $\mathbf{y}$ represents the activity of output neurons, τ_{out} denotes the output neurons' time constant, W^{out} signifies the synaptic weight matrix connecting recurrent and output neurons, b^{out} is the bias term, and $\mathbf{r}^{\text{rec}}$ concatenates both membrane potentials and spike activities of the recurrent neurons. For computational implementations, a discrete-time approximation of this dynamics is employed:

$$\mathbf{y}^t = e^{-\Delta t/\tau_{\text{out}}} \mathbf{y}^{t-1} + (1 - e^{-\Delta t/\tau_{\text{out}}})(W^{\text{out}} \mathbf{r}^{\text{rec},t-1} + b^{\text{out}}). \tag{77}$$

7.8.3 Weight initialization

For the networks described in Eqs. 43-51, we employed a specific weight initialization strategy for input, recurrent, and readout connections. The weights were sampled from a scaled Gaussian distribution, defined as: $W_{ji} \sim \sqrt{\frac{s}{N_{\text{in}}}}\mathcal{N}(0,1)$, where W_{ji} represents the weight from neuron i to neuron j, N_{in} denotes the number of afferent (input-providing) neurons, $\mathcal{N}(0,1)$ is the standard normal distribution with mean 0 and variance 1, s is a scaling factor that allows for control over the initial magnitude of the weights. This initialization schema helps maintain consistent variance of activations and gradients across layers, potentially aiding in training stability and convergence.

For the network with E/I separation, as described in Eq. 52, we modified the initialization procedure slightly. While still using the scaled Gaussian distribution, we applied an absolute value operation to ensure all weights were non-negative: $W_{ji} \sim \sqrt{\frac{s}{N_{\text{in}}}}|\mathcal{N}(0,1)|$. This modification ensures that excitatory remain excitatory and inhibitory connections remain inhibitory, adhering to Dale's principle often observed in biological networks. The absolute value operation maintains the scale and distribution shape while restricting weights to positive values.

7.8.4 Delayed match-to-sample task

We applied our learning algorithms to train a recurrent spiking network (Eq. 46) on a delayed match-to-sample (DMTS) task [14, 16] to test the long-term dependency learning capability of our learning algorithms (see Section 5.1). This task comprises four phases: fixation, sample, delay, and test (Figure S8). The network includes 100 motion neurons, each covering a 360-degree range of directions. During the fixation phase, the network dynamics are initialized, and motion neurons fire at the baseline rate due to the absence of stimuli. In the sample and test phases, one of eight motion directions is presented to the motion neurons and lasts 500 ms. The delay phase, lasting over 1000 ms, occurs between the sample and test phases, during which motion neurons revert to the baseline firing rate. This task is challenging since it requires the network to maintain the sample direction information over a long delay period and then make a decision based on the test stimulus. The decision is made during the test phase, where network activity is processed by a leaky readout layer to determine whether the sequentially presented directions match or not.

In the DMTS task, the input to the network is provided by 100 motion-direction-tuned neurons. These neurons are designed to simulate the response properties of direction-selective cells found in visual cortical areas such as the middle temporal area. The tuning of these motion direction-selective neurons follows a von Mises distribution, which is a circular analog of the Gaussian distribution appropriate for directional data. The firing rate activity u_i^t of input neuron i during the stimulus presentation period is given by:

$$u_i^t = A \exp\left(\kappa \cos\left(\theta - \theta_{\text{pref}}^i\right)\right) \tag{78}$$

where θ is the direction of the presented stimulus, θ_{pref}^i is the preferred direction of input neuron i, κ is the concentration parameter, which is set to 2.0 in this study, and A is the maximum firing rate, which is set to 40 Hz.

The von Mises distribution provides a peak response when the stimulus direction θ matches the neuron's preferred direction θ_{pref}^i, with a smooth fall-off for other directions. The concentration parameter κ controls the width of this tuning curve, with higher values resulting in sharper tuning.

In addition to their direction-tuned responses during stimulus presentation, these input neurons maintain a constant background firing rate r_{bg} during all periods of the task, including inter-stimulus intervals. This background rate is set to 1 Hz, simulating the spontaneous activity observed in cortical neurons even in the absence of specific stimuli.

Finally, we generated the spike train of motion direction-tuned neurons using a Poisson process with the above-calculated firing rate $u_i^t + r_{\text{bg}}$ during sample and test periods. Two examples were shown in Figure S8.

7.8.5 Evidence accumulation task

The evidence accumulation paradigm is a fundamental tool in neuroscience and cognitive psychology for exploring the mechanisms of decision-making. Our study utilizes a task design similar to the

one described by Morcos et al. [39], where a virtual-reality T-maze assesses the decision-making capabilities of a head-restrained mouse. As the mouse navigates the maze, it encounters visual cues on both the left and right sides, as illustrated in Figure 6A. Upon reaching the T-junction, the mouse must choose a direction based on the majority of cues received from one side, regardless of the sequence or the position of the last cue. The complexity of the task lies in the requirement for the mouse to count and compare cues from each side independently and retain this information until the decision point, well before any reward is offered.

To simulate this task, we employed a population consisting of 100 neurons segmented into four groups. The first and second groups, each comprising 25 neurons, encode the cues from the left and right sides, respectively. Another 25 neurons process the recall cue during a subsequent recall period, and the final group of 25 neurons generates constant background noise at a rate of 10 Hz. The cues and the recall stimuli are modeled with Poisson noise at a firing rate of 40 Hz, with each stimulus lasting 150 ms and interspersed with 50 ms intervals of silence. The recall stimulus follows a delay of 1000 ms, during which background noise persists. The operational details of this neural representation are further depicted in Figure 6A.

7.9 Robust Automatic Online Learning through the Jaxpr Compilation

The backpropagation algorithm, a landmark in machine learning, has gained prominence due to its computational efficiency and seamless integration with modern automatic differentiation frameworks like PyTorch [52] and TensorFlow [1]. These frameworks simplify the implementation of backpropagation, allowing researchers and developers to focus on innovating and optimizing model architectures without delving into underlying details. Automatic differentiation systems efficiently and accurately perform gradient computations, further enhancing the algorithm's utility.

Inspired by the success of backpropagation, we aim to develop a general, scalable programming interface for online learning algorithms in SNNs. This interface allows to lower technical barriers by abstracting complex learning mechanisms, such as the automatic derivation and generation of eligibility traces and learning signals. Users need only define the network's dynamics and architecture, while the interface manages the subsequent online learning processes, including gradient calculations and parameter updates. Our design philosophy promotes the widespread adoption of SNN online learning algorithms across diverse model dynamics and architectures.

However, the development of such a framework presents three core challenges: (1) deriving eligibility traces, (2) generating learning signals, and (3) computing parameter gradients. To address these challenges, we leverage the Jaxpr language, an intermediate representation in JAX [11], which facilitates the inspection of computation graphs using abstract symbols. Without Jaxpr, analyzing the mathematical equations of user-defined models would be difficult. By abstracting user-defined Python code into manipulable symbols, we can derive and generate the necessary components for our online learning algorithms. Ultimately, this allows us to regenerate the Jaxpr code, encompassing both hidden state evolution and the online learning algorithms. We refer to this process as *online learning compilation*.

Deriving eligibility traces. For the automatic derivation of *eligibility traces*, our compilation consists of three steps.

The first step involves analyzing the hidden state group $\mathbf{h}_l$ that belong to a single recurrent layer. We aim to understand which state variables compose the hidden state $\mathbf{h}_l = [\mathbf{v}_l^1, \cdots, \mathbf{v}_l^d]$ (where d is the number of variables in the hidden states), and how these state variables interact and transition into each other. This analysis is essential for computing the block diagonal matrix $\mathbf{D}^t = \begin{pmatrix} \partial \mathbf{v}_l^1 / \partial \mathbf{v}_l^1 & \cdots & \partial \mathbf{v}_l^1 / \partial \mathbf{v}_l^d \\ \vdots & \ddots & \vdots \\ \partial \mathbf{v}_l^d / \partial \mathbf{v}_l^1 & \cdots & \partial \mathbf{v}_l^d / \partial \mathbf{v}_l^d \end{pmatrix}$. This step yields multiple hidden state groups $[\mathbf{h}_1, \cdots \mathbf{h}_l, \cdots, \mathbf{h}_L]$, and their corresponding block diagonal matrices $\mathbf{D}^t$.

The second step involves analyzing how each weight parameter is connected to hidden state groups, requiring us to compute the weight-to-hidden relationships. By systematically enumerating the computation graph, we inspect which state variable $\mathbf{v}_l^j$ each parameter $\boldsymbol{\theta}_k$ is connected to, where l is the index of the hidden state group, j is the index of the variable in the hidden group $\mathbf{h}_l$, and k is the index of the weight parameter. We then establish the relationship between the parameter $\boldsymbol{\theta}_k$ and

the hidden state group $\mathbf{h}_l$. These dependencies are crucial for computing the block diagonal matrix $\mathbf{D}_f^t = \begin{pmatrix} \partial \mathbf{v}_l^1 / \partial \boldsymbol{\theta} k \\ \vdots \\ \partial \mathbf{v}_l^d / \partial \boldsymbol{\theta}_k \end{pmatrix}$.

The third step involves generating the Jaxpr representation of eligibility traces based on the analysis results from the previous two steps. This process is crucial for translating our analytical insights into executable code. We leverage the symbolic Jaxpr language to generate mathematical operations as presented in Eqs. 3-6.

Generating learning signals. To generate the *learning signals* essential for Eq. 7, we currently employ the standard backpropagation algorithm through deep layers to maximize learning performance. This approach has proven effective in various deep-learning applications, allowing for the efficient propagation of error signals from the readout layer back through the hidden states.

However, implementing backpropagation through multiple hidden layers presents a significant challenge in the JAX framework. Specifically, it necessitates computing derivatives of gradients with respect to intermediate variables $[\mathbf{h}_1, \cdots \mathbf{h}_l, \cdots, \mathbf{h}_L]$. This operation is particularly difficult in JAX due to its inability to save gradients of intermediate variables. While JAX excels at gradient computation for explicit function parameters, hidden states computed within a function cannot be directly passed as function parameters, creating a bottleneck in the backpropagation process.

To overcome this limitation, we employ a perturbation function. Our approach involves adding a zero perturbation $\delta_{\mathbf{h}}$ to the computed hidden states, resulting in $\mathbf{h}^t + \delta_{\mathbf{h}}$. By treating this zero perturbation as a function parameter, we can effectively compute the required derivatives. This method capitalizes on the fact that the derivatives of the zero perturbation $\frac{\partial \mathcal{L}^t}{\partial \delta_{\mathbf{h}^t}}$ are mathematically equivalent to the derivatives of the corresponding hidden states $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}$.

The practical implementation of this technique involves several key steps. We begin by rewriting the model's computation graph, incorporating a perturbation term into each hidden state: $\mathbf{h}^t \leftarrow \mathbf{h}^t + \delta_{\mathbf{h}^t}$. This modification allows us to introduce the perturbation without altering the network results. Then, utilizing the `jax.vjp` (vector-Jacobian product) interface provided by JAX, we compute the derivatives of the zero perturbations $\frac{\partial \mathcal{L}^t}{\partial \delta_{\mathbf{h}^t}}$. Finally, we extract this computation as a Jaxpr representation, allowing it to be leveraged in subsequent steps.

Computing parameter gradients. In the final stage of our algorithm, we integrate the code for eligibility traces and learning signals generated in the preceding steps to compute parameter gradients. To maintain consistency with established automatic differentiation routines, we leverage JAX's `jax.custom_vjp` functionality to register the parameter gradient computation through a customized vector-Jacobian product (VJP) rule. This approach allows us to seamlessly incorporate our online gradient computation method into JAX's existing differentiation framework.

Our implementation revolves around the definition of two critical functions:

1. Forward Pass Function: This function encapsulates the forward propagation of the network and the computation of eligibility traces by merging the Jaxpr representation of two steps.
2. Backward Pass Function: This function is responsible for: a) Computing the learning signals $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}$ based on the provided learning targets. b) Calculating the parameter gradients by combining the eligibility traces delivered from the forward pass and learning signals.

The integration of these two functions within the `jax.custom_vjp` interface allows us to take online gradient computation through the established JAX's autograd routines, such like `jax.grad` or `jax.vjp` functions. See Listing S3 for an example of *BrainScale*'s online gradient computation.

Overall, the out-of-the-box interface for SNN online learning in *BrainScale* is remarkably simple, requiring no more than two lines of code. For example:

```
# define online computation using the ES-D-RTRL algorithm
model = brainscale.DiagIODimAlgorithm(net, decay_or_rank=0.98)

# define online computation using the D-RTRL algorithm
```

```
model = nn.DiagParamDimAlgorithm(net)

# compile the online learning graph using the input data
model.compile_graph(inputs)
```

Data availability

The datasets used in this study are open source and publicly available. The Neuromorphic-MNIST dataset is freely available at https://www.garrickorchard.com/datasets/n-mnist. The Spiking Heidelberg Digits (SHD) dataset is publicly available at https://zenkelab.org/resources/spiking-heidelberg-datasets-shd. The IBM DVS Gesture dataset can be downloaded from https://ibm.ent.box.com/s/3hiq58ww1pbbjrinh367ykfdf60xsfm8/folder/50167556794. The DMTS and evidence accumulation tasks are generated in this study, and can be found in the public available GitHub repository (https://github.com/chaoming0625/brainscale-exp-for-snns, which will be made available upon the acceptance of the manuscript).

Code availability

Upon the acceptance of the manuscript, *BrainScale* will be distributed via the pypi package index (https://pypi.org/project/brainscale/) and publicly released on GitHub (https://github.com/chaoming0625/brainscale) under the license of Apache License v2.0. Its documentation will be hosted on the free documentation hosting platform Read the Docs (https://brainscale.readthedocs.io/). *BrainScale* can be used in Windows, macOS, and Linux operating systems. The code to reproduce the experimental evaluations in this study will be publicly available from the following GitHub repository (https://github.com/chaoming0625/brainscale-exp-for-snns).

Acknowledgments

This work was supported by the Science and Technology Innovation 2030-Brain Science and Brain-inspired Intelligence Project (No. 2021ZD0200204, SW), the Young Scientists Fund of the National Natural Science Foundation of China (No. 3240070449, CMW), the China Postdoctoral Science Foundation (No. 2024M750076, CMW), and the Postdoctoral Fellowship Program of CPSF (No. GZC20230106, CMW).

Competing interests

The authors declare that they have no competing interests.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016.

[2] Larry F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). Brain research bulletin, 50(5-6):303–304, 1999.

[3] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7243–7252, 2017.

[4] David Attwell and Simon B Laughlin. An energy budget for signaling in the grey matter of the brain. Journal of Cerebral Blood Flow & Metabolism, 21(10):1133–1145, 2001.

[5] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. Advances in neural information processing systems, 31, 2018.

[6] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. Nature communications, 11(1):3625, 2020.

[7] Yazan N Billeh, Binghuang Cai, Sergey L Gratiy, Kael Dai, Ramakrishnan Iyer, Nathan W Gouwens, Reza Abbasi-Asl, Xiaoxuan Jia, Joshua H Siegle, Shawn R Olsen, et al. Systematic integration of structural and functional data into multi-scale models of mouse primary visual cortex. Neuron, 106(3):388–403, 2020.

[8] Thomas Bohnstingl, Stanisław Woźniak, Angeliki Pantazi, and Evangelos Eleftheriou. Online spatio-temporal learning in deep neural networks. IEEE Transactions on Neural Networks and Learning Systems, 2022.

[9] Jacopo Bono and Claudia Clopath. Modeling somatic and dendritic spike mediated plasticity at the single neuron and network level. Nature communications, 8(1):706, 2017.

[10] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. 2018.

[11] 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.

[12] Romain Brette and Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. Journal of neurophysiology, 94(5):3637–3642, 2005.

[13] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

[14] Y Chudasama. Delayed (non) match-to-sample task. Encyclopedia of Psychopharmacology, pages 372–372, 2010.

[15] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. IEEE Transactions on Neural Networks and Learning Systems, 2020.

[16] Thomas A Daniel, Jeffrey S Katz, and Jennifer L Robinson. Delayed match-to-sample in working memory: A brainmap meta-analysis. Biological Psychology, 120:10–20, 2016.

[17] Egidio D'Angelo and Viktor Jirsa. The quest for multiscale brain modeling. Trends in neurosciences, 2022.

[18] Nicolas Fourcaud-Trocmé, David Hansel, Carl Van Vreeswijk, and Nicolas Brunel. How spike generation mechanisms determine the neuronal response to fluctuating inputs. Journal of neuroscience, 23(37):11628–11640, 2003.

[19] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. Frontiers in neural circuits, 9:85, 2016.

[20] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. Neuronal dynamics: From single neurons to networks and models of cognition. Cambridge University Press, 2014.

[21] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752, 2023.

[22] Bing Han and Kaushik Roy. Deep spiking neural network: Energy efficiency through time based coding. In European conference on computer vision, pages 388–404. Springer, 2020.

[23] Michael Hausser, Nelson Spruston, and Greg J Stuart. Diversity and dynamics of dendritic signaling. Science, 290(5492):739–744, 2000.

[24] 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.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[26] Eugene M Izhikevich. Simple model of spiking neurons. IEEE Transactions on neural networks, 14(6):1569–1572, 2003.

[27] Eugene M Izhikevich. Which model to use for cortical spiking neurons? IEEE transactions on neural networks, 15(5):1063–1070, 2004.

[28] Patrick Kaifosh and Attila Losonczy. Mnemonic functions for nonlinear dendritic integration in hippocampal pyramidal circuits. Neuron, 90(3):622–634, 2016.

[29] Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, A James Hudspeth, Sarah Mack, et al. Principles of neural science, volume 4. McGraw-hill New York, 2000.

[30] George Kastellakis, Alcino J Silva, and Panayiota Poirazi. Linking memories across time via neuronal and dendritic overlaps in model neurons with active dendrites. Cell reports, 17(6):1491–1504, 2016.

[31] Peter Kloppenburg and Martin Paul Nawrot. Neural coding: sparse but on time. Current Biology, 24(19):R957–R959, 2014.

[32] Peter E Latham, BJ Richmond, PG Nelson, and S Nirenberg. Intrinsic dynamics in neuronal networks. i. theory. Journal of neurophysiology, 83(2):808–827, 2000.

[33] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. Nature communications, 7(1):13276, 2016.

[34] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. Nature Reviews Neuroscience, 21(6):335–346, 2020.

[35] Michael London and Michael Häusser. Dendritic computation. Annu. Rev. Neurosci., 28(1):503–532, 2005.

[36] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. Neural networks, 10(9):1659–1671, 1997.

[37] Nikola T Markov, Maria M Ercsey-Ravasz, AR Ribeiro Gomes, Camille Lamy, Loic Magrou, Julien Vezoli, Pierre Misery, Arnaud Falchier, Rene Quilodran, Marie-Alice Gariel, et al. A weighted and directed interareal connectivity matrix for macaque cerebral cortex. Cerebral cortex, 24(1):17–36, 2014.

[38] Adnan Mehonic and Anthony J Kenyon. Brain-inspired computing needs a master plan. Nature, 604(7905):255–260, 2022.

[39] Ari S Morcos and Christopher D Harvey. History-dependent variability in population dynamics during evidence accumulation in cortex. Nature neuroscience, 19(12):1672–1681, 2016.

[40] 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):eaay3134, 2019.

[41] Michael C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. Complex Syst., 3, 1989.

[42] Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random kronecker factors. Advances in Neural Information Processing Systems, 31, 2018.

[43] James M Murray. Local online learning in recurrent networks with random feedback. Elife, 8:e43299, 2019.

[44] Farzan Nadim and Dirk Bucher. Neuromodulation of neurons and synapses. Current opinion in neurobiology, 29:48–56, 2014.

[45] Richard Naud and Henning Sprekeler. Sparse bursts optimize information transmission in a multiplexed neural code. Proceedings of the National Academy of Sciences, 115(27):E6329–E6338, 2018.

[46] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. IEEE Signal Processing Magazine, 36(6):51–63, 2019.

[47] Bruno A Olshausen and David J Field. Sparse coding of sensory inputs. Current opinion in neurobiology, 14(4):481–487, 2004.

[48] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. Frontiers in neuroscience, 9:437, 2015.

[49] Thomas Ortner, Lorenzo Pes, Joris Gentinetta, Charlotte Frenkel, and Angeliki Pantazi. Online spatio-temporal learning with target projection. In 2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS), pages 1–5. IEEE, 2023.

[50] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In International Conference on Machine Learning, pages 26670–26698. PMLR, 2023.

[51] Michalis Pagkalos, Spyridon Chavlis, and Panayiota Poirazi. Introducing the dendrify framework for incorporating dendrites to spiking neural networks. Nature Communications, 14(1):131, 2023.

[52] 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.

[53] Nicolas Perez-Nieves, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. Neural heterogeneity promotes robust learning. Nature communications, 12(1):5791, 2021.

[54] Panayiota Poirazi and Athanasia Papoutsi. Illuminating dendritic function with computational models. Nature reviews neuroscience, 21(6):303–321, 2020.

[55] Fernando M Quintana, Fernando Perez-Peña, Pedro L Galindo, Emre O Neftci, Elisabetta Chicca, and Lyes Khacef. Etlp: Event-based three-factor local plasticity for online learning with neuromorphic hardware. Neuromorphic Computing and Engineering, 4(3):034006, 2024.

[56] Wade G Regehr. Short-term presynaptic plasticity. Cold Spring Harbor perspectives in biology, 4(7):a005702, 2012.

[57] Mengye Ren, Simon Kornblith, Renjie Liao, and Geoffrey Hinton. Scaling forward gradient with local losses. arXiv preprint arXiv:2210.03310, 2022.

[58] Anthony J Robinson and Frank Fallside. The utility driven dynamic error propagation network, volume 1. University of Cambridge Department of Engineering Cambridge, 1987.

[59] Julian Rossbroich, Julia Gygax, and Friedemann Zenke. Fluctuation-driven initialization for spiking neural network training. Neuromorphic Computing and Engineering, 2(4):044016, 2022.

[60] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. Nature, 575(7784):607–617, 2019.

[61] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. nature, 323(6088):533–536, 1986.

[62] Caroline A Runyan, Eugenio Piasini, Stefano Panzeri, and Christopher D Harvey. Distinct timescales of population coding across cortex. Nature, 548(7665):92–96, 2017.

[63] Maximilian Schmidt, Rembrandt Bakker, Kelly Shen, Gleb Bezgin, Markus Diesmann, and Sacha Jennifer van Albada. A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas. PLOS Computational Biology, 14(10):e1006359, 2018.

[64] Catherine D Schuman, Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, Bill Kay, et al. Opportunities for neuromorphic computing algorithms and applications. Nature Computational Science, 2(1):10–19, 2022.

[65] Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. Advances in neural information processing systems, 31, 2018.

[66] Sen Song, Kenneth D Miller, and Larry F Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. Nature neuroscience, 3(9):919–926, 2000.

[67] Nelson Spruston. Pyramidal neurons: dendritic structure and synaptic integration. Nature Reviews Neuroscience, 9(3):206–221, 2008.

[68] Mototaka Suzuki, Cyriel MA Pennartz, and Jaan Aru. How deep is the brain? the shallow brain hypothesis. Nature Reviews Neuroscience, 24(12):778–791, 2023.

[69] Panagiota Theodoni, Piotr Majka, David H Reser, Daniel K Wójcik, Marcello GP Rosa, and Xiao-Jing Wang. Structural attributes and principles of the neocortical connectome in the marmoset monkey. Cerebral Cortex, 32(1):15–28, 2022.

[70] Misha Tsodyks, Klaus Pawelzik, and Henry Markram. Neural networks with dynamic synapses. Neural computation, 10(4):821–835, 1998.

[71] Misha Tsodyks and Si Wu. Short-term synaptic plasticity. Scholarpedia, 8(10):3153, 2013.

[72] Alexandra Tzilivaki, George Kastellakis, and Panayiota Poirazi. Challenging the point neuron dogma: Fs basket cells as 2-stage nonlinear integrators. Nature communications, 10(1):3664, 2019.

[73] Rufin VanRullen, Rudy Guyonneau, and Simon J Thorpe. Spike times make sense. Trends in neurosciences, 28(1):1–4, 2005.

[74] Tim P Vogels and Larry F Abbott. Signal propagation and logic gating in networks of integrate-and-fire neurons. Journal of neuroscience, 25(46):10786–10795, 2005.

[75] 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:e86365, 2023.

[76] 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:e86365, dec 2023.

[77] Chaoming Wang, Tianqiu Zhang, Sichao He, Hongyaoxing Gu, Shangyang Li, and Si Wu. A differentiable brain simulator bridging brain simulation and brain-inspired computing. In The Twelfth International Conference on Learning Representations, 2024.

[78] Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. Neural Networks, 1:339–356, 1988.

[79] Ronald J. Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. Neural Computation, 2:490–501, 1990.

[80] Ronald J Williams and David Zipser. Experimental analysis of the real-time recurrent learning algorithm. Connection science, 1(1):87–111, 1989.

[81] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. Neural computation, 1(2):270–280, 1989.

[82] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Online training through time for spiking neural networks. Advances in Neural Information Processing Systems, 35:20717–20730, 2022.

[83] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara, and Ngan Le. Spiking neural networks and their applications: A review. Brain Sciences, 12(7):863, 2022.

[84] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. Neural computation, 30(6):1514–1541, 2018.

[85] Shimin Zhang, Qu Yang, Chenxiang Ma, Jibin Wu, Haizhou Li, and Kay Chen Tan. Tc-lif: A two-compartment spiking neuron model for long-term sequential modelling. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pages 16838–16847, 2024.

[86] Hanle Zheng, Zhong Zheng, Rui Hu, Bo Xiao, Yujie Wu, Fangwen Yu, Xue Liu, Guoqi Li, and Lei Deng. Temporal dendritic heterogeneity incorporated with spiking neural networks for learning multi-timescale dynamics. Nature Communications, 15(1):277, 2024.

Supplementary Information

Chaoming Wang[1,2], **Xingsi Dong**[1,2,3], **Jiedong Jiang**[4], **Zilong Ji**[5], **Xiao Liu**[1,2], **Si Wu**[1,2,3,*]
[1] School of Psychological and Cognitive Sciences, Peking University, Beijing, China
[2] PKU-IDG/McGovern Institute for Brain Research, Peking University, Beijing, China
[3] Center of Quantitative Biology, Peking-Tsinghua Center for Life Sciences,
Academy for Advanced Interdisciplinary Studies, Peking University, Beijing, China
[4] Beijing International Center for Mathematical Research, Peking University, Beijing, China
[5] Institute of Cognitive Neuroscience, University College London, London, United Kingdom
[*] Corresponding author: `siwu@pku.edu.cn`

Contents

A Approximation Accuracy on Weight Gradients **1**

B RNNs as a Special Form of Brain Dynamics Models **2**

C Code Examples of *BrainScale* Programming Interface **3**

D Supplementary Figures **6**

E Supplementary Tables **20**

A Approximation Accuracy on Weight Gradients

We explored factors affecting the performance of the hidden Jacobian and weight gradient estimation in our online learning algorithms.

One prominent factor might be the network size, which determines the scalability of our proposed learning algorithms for large-scale SNN networks. Our evaluations demonstrated that the size of recurrent units did no significantly influence the Jacobian and gradient approximation, regardless of the evaluated dataset or network dynamics (Figure S12). This result indicates that our online learning rules can be effectively applied to larger SNN models. Remarkably, for the ES-D-RTRL algorithm, we even observed a slight increase in gradient cosine similarity as the network size increased.

Another factor we explored was the firing rate, which is determined by the feedforward and recurrent connections. By increasing the feedforward connection strength, we observed an increase in firing rate accompanied by a slight decrease in cosine similarity for both Jacobian and gradient estimation (Figure S13). However, for the ES-D-RTRL algorithm, the gradient cosine similarity consistently increased, indicating a preference for higher firing rates. Notably, the strength of recurrent connections had a more pronounced effect than feedforward connections. Increasing the strength of recurrent connections resulted in substantially lower cosine similarity (Figure S14). This effect can be attributed to the increase in both firing frequency and the size of non-zero elements in the non-zero columns of the Jacobian matrix caused by the recurrent spiking connections.

Finally, we examined the depeth of recurrent layers on weight gradients. We assessed the layer-wise cosine similarity between online gradient estimates and offline BPTT gradients using the SHD dataset [4]. Specifically, we evaluated a five-layer SNN network with 200 hidden units per layer, considering a variety of neuronal and synaptic dynamics. Consistently, we observed a decrease in cosine similarity as the layer depth increased (see Figure S15). Lower layers exhibited lower cosine similarity to its their exact gradients, corresponding to the fact that more recurrence information of hidden states in top layers is ignored.

B RNNs as a Special Form of Brain Dynamics Models

Recurrent neural networks have been widely applied in modeling sequence data in various domains, such as language modeling [10, 14, 6], translation [3], and speech recognition [5]. Among them, Long Short-Term Memory Networks (LSTM) [8] and Gated Recurrent Units (GRU) [3] are the most prominent model architectures. Their success can be largely due to the incorporation of element-wise linear gated recurrence. Decades of research on gated RNNs [8, 3, 2, 7, 16, 15, 14, 6], which have consistently shown that preserving long-range dependencies in RNNs requires the incorporation of element-wise linear recurrence. For instance, in a classical gated RNN, GRU [3], the hidden state $\mathbf{h}$ is updated as:

$$\mathbf{h}^t = \mathbf{f}^t \circ \mathbf{h}^{t-1} + (1 - \mathbf{f}^t) \circ \mathbf{x}^t, \tag{S79}$$

where $\mathbf{f}^t$ is the update gate (typically close to 0 or 1) and $\mathbf{x}^t$ is the encoded input. When the gate $\mathbf{f}$ approaches 1, the network can propagate information over a longer time sequence. Moreover, the hidden Jacobian $\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = D_{\mathbf{f}^t} + \cdots$ remains close to $\mathbf{I}$ during training, allowing any error vector to backpropagate through the hidden state over long time steps. Here, $D_{\mathbf{f}^t}$ denotes a diagonal matrix with vector $\mathbf{f}^t$ as its diagonal entries, and $\mathbf{I}$ represents the identity matrix.

The element-wise gated recurrence in RNNs can be viewed as a simplified form of complex brain simulation dynamics. Consider a basic implementation, $\tau \mathrm{d}\mathbf{h}/\mathrm{d}t = -\mathbf{h} + \mathbf{I}$. Using exponential Euler integration, this neural dynamics can be discretized as:

$$\mathbf{h}^t = \exp(-\Delta t/\tau) \circ \mathbf{h}^{t-1} + \lambda(1 - \exp(-\Delta t/\tau)) \circ \mathbf{I}^t, \tag{S80}$$

where λ is a scaling factor. Eq. S80 is equivalent to Eq. S79 when τ is a dynamic time constant.

C Code Examples of *BrainScale* Programming Interface

To illustrate the capabilities of the *BrainScale* framework, we present three exemplary cases of model customization and gradient computation.

First, we demonstrate the implementation of a dense connection layer (also known as an MLP layer) using the `ETraceParamOp` concept (Listing S1).

Second, we showcase the definition of a Leaky Integrate-and-Fire (LIF) neuron model, leveraging the `ETraceVar` concept (Listing S2).

Finally, we illustrate the integration of these two models to construct a spiking network and compute the gradient of network parameters based on specified learning targets (Listing S3).

These examples underscore *BrainScale*'s user-friendly interface, adaptability, and expandability, empowering users to effortlessly customize, define, and train their own models.

```python
import jax.numpy as jnp
import brainstate as bst
import brainscale as nn

class Linear(bst.nn.DnnLayer):
  def __init__(self, n_in, n_out,
               w_init=bst.init.KaimingNormal(),
               b_init=bst.init.ZeroInit()):
    super().__init__()
    self.in_size = (n_in,)
    self.out_size = (n_out,)

    # parameters
    param = dict(weight=w_init([n_in, n_out]), bias=b_init([n_out]))
    # operations
    op = lambda x, w: jnp.dot(x, w['weight']) + w['bias']
    # the etrace weight and op
    self.weight_op = nn.ETraceParamOp(param, op)

  def update(self, x):
    # call the model by ".execute" the weight_op
    return self.weight_op.execute(x)
```

Listing S1: Defining a dense connection layer in *BrainScale*.

```python
import brainscale as nn
import brainstate as bst
import jax

class LIF(bst.nn.Neuron):
  def __init__(
      self, in_size, keep_size=False, tau=5.,
      V_th=1., V_reset=0., V_rest=0.,
      spk_fun=bst.surrogate.ReluGrad(), spk_reset='soft'
  ):
    super().__init__(in_size, keep_size=keep_size,
                     spk_fun=spk_fun, spk_reset=spk_reset)

    # parameters
    self.tau = tau
    self.V_th = V_th
    self.V_rest = V_rest
    self.V_reset = V_reset

  def dv(self, v, t, x):
    # "sum_current_inputs()" sums up all incoming currents
    x = self.sum_current_inputs(v, init=x)
    # the differential equation
    return (-v + self.V_rest + x) / self.tau
```

```python

  def init_state(self, batch_size: int = None, **kwargs):
    # initialize the membrane potential
    bs = () if batch_size is None else (batch_size,)
    V = jax.numpy.full(bs + self.varshape, self.V_rest)
    self.V = nn.ETraceVar(V)

  def get_spike(self, V):
    # scale the membrane potential
    v_scaled = (V - self.V_th) / self.V_th
    # generate spike using the surrogate gradient function
    return self.spk_fun(v_scaled)

  def update(self, x=0.):
    # the last spike and membrane potential
    last_v = self.V.value
    lst_spk = self.get_spike(last_v)
    if self.spk_reset == 'soft':
       V_th = self.V_th
    else:
       V_th = jax.lax.stop_gradient(last_v)
    V = last_v - (V_th - self.V_reset) * lst_spk
    # the current membrane potential
    V = bst.nn.exp_euler_step(self.dv, V, None, x)
    V = V + self.sum_delta_inputs()
    self.V.value = V
    # the current spike
    return self.get_spike(V)
```

Listing S2: Defining a LIF neuron model in *BrainScale*.

```python
import brainscale as nn
import brainstate as bst
import braintools as bts
import jax
import jax.numpy as jnp


class LIF_Delta_Net(bst.Module):
   """
   The network with LIF neurons and dense connected with delta
    synapses.
   """

   def __init__(
      self, n_in, n_rec, n_out, tau_mem=5., V_th=1.,
   ):
      super().__init__()

      # using "LIF" defined above to construct
      # a neuron population with LIF dynamics
      self.neu = LIF(n_rec, tau=tau_mem, V_th=V_th)

      # using "Linear" defined above to
      # construct a delta synapse projection
      self.syn = bst.nn.HalfProjDelta(
          Linear(n_in + n_rec, n_rec),
          self.neu
      )

      # the readout layer
      self.readout = nn.LeakyRateReadout(n_rec, n_out)

   def update(self, spk):
      spk = jnp.concat([spk, self.neu.get_spike()], axis=-1)
```

```python
        self.syn(spk)
        return self.readout(self.neu())


# define the one-batch inputs and targets
n_seq, n_batch = 512, 128
inputs = bst.random.rand(n_seq, n_batch, 10) < 0.1
targets = bst.random.randint(0, 2, (n_batch,))

# instantiate a spiking network
net = LIF_Delta_Net(10, 100, 2)
bst.init_states(net, batch_size=n_batch)

# online learning algorithm
method = 'es-diag'  # or 'diag' or 'hybrid'
if method == 'es-diag':  # ES-D-RTRL algorithm
  model = nn.DiagIODimAlgorithm(net, decay_or_rank=0.98)

elif method == 'diag':  # D-RTRL algorithm
  model = nn.DiagParamDimAlgorithm(net)

elif method == 'hybrid':  # hybrid algorithm
  model = nn.DiagHybridDimAlgorithm(net, decay_or_rank=0.98)

else:
  raise ValueError(f'Unknown online learning methods: {method}.')

# compile the eligibility trace graph using the one-step input
model.compile_graph(inputs[0])

# retrieve parameters that need to compute gradients
weights = net.states().subset(bst.ParamState)


# define loss function
def loss_fn(i, x):
  out = model(x, running_index=i)
  return jnp.mean(bts.metric.softmax_cross_entropy_with_integer_labels
    (out, targets))


# gradient computation using traditional autograd interface
def step_grad(last_grads, ix):
  # gradients computed at the current step
  grads = bst.transform.grad(loss_fn, weights)(ix[0], ix[1])
  # accumulate gradients: prev + current
  new_grads = jax.tree.map(jnp.add, last_grads, grads)
  return new_grads, None


# loop over all sequences to compute gradients
indices = jnp.arange(n_seq)
init_grads = jax.tree.map(jnp.zeros_like, weights.to_dict_values())
grads = bst.transform.scan(step_grad, init_grads, (indices, inputs))
```

Listing S3: Defining a network in *BrainScale* in which LIF neurons (defined in Listing S2) are interconnected with Delta synapses (see the linear transformation in Listing S1).

D Supplementary Figures

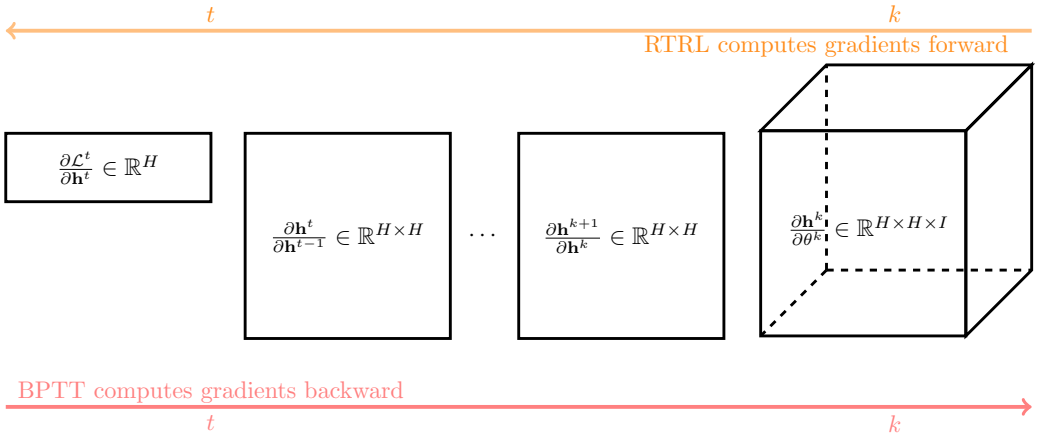

Figure S7: **Comparison of Gradient Computation Methods: BPTT vs. RTRL**. This figure illustrates the differences between BPTT and RTRL in computing gradients $\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}^k}$ for $t \geq k$. **BPTT**: Gradients are computed in a backward manner. The hidden states $\mathbf{h}$ are stored from time steps k to t. The gradient $\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}^k}$ is then propagated backward through time using the chain rule $\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}^k} = \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} \cdots \frac{\partial \mathbf{h}^k}{\partial \boldsymbol{\theta}^k}$. **RTRL**: Gradients are computed in a forward manner. At time step k, the influence Jacobian $\boldsymbol{\epsilon}^k = \frac{\partial \mathbf{h}^k}{\partial \boldsymbol{\theta}^k}$ is calculated. At the next time step $k+1$, the influence Jacobian is updated to $\boldsymbol{\epsilon}^{k+1} = \frac{\partial \mathbf{h}^{k+1}}{\partial \mathbf{h}^k} \boldsymbol{\epsilon}^k$. This process continues forward in time, with $\boldsymbol{\epsilon}^{k+2} = \frac{\partial \mathbf{h}^{k+2}}{\partial \mathbf{h}^{k+1}} \boldsymbol{\epsilon}^{k+1}$, and so forth.

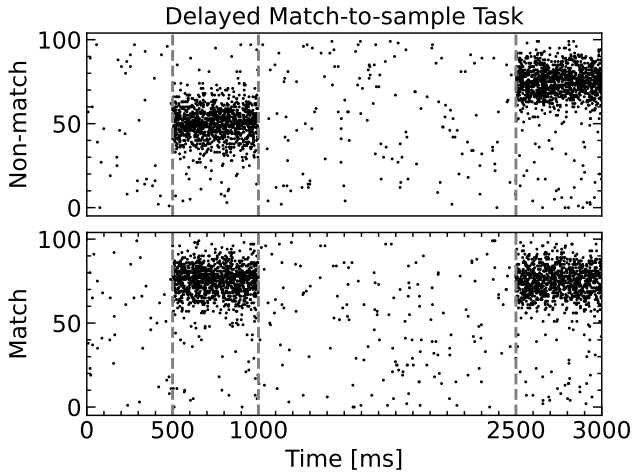

Figure S8: **Spiking Dynamics of Input Neurons During the DMTS Task: Two Examples.** The top panel illustrates the non-match case, while the bottom panel shows the match case. The task consists of four sequential phases: fixation (500 ms), sample (500 ms), delay (1000 ms), and test (500 ms).

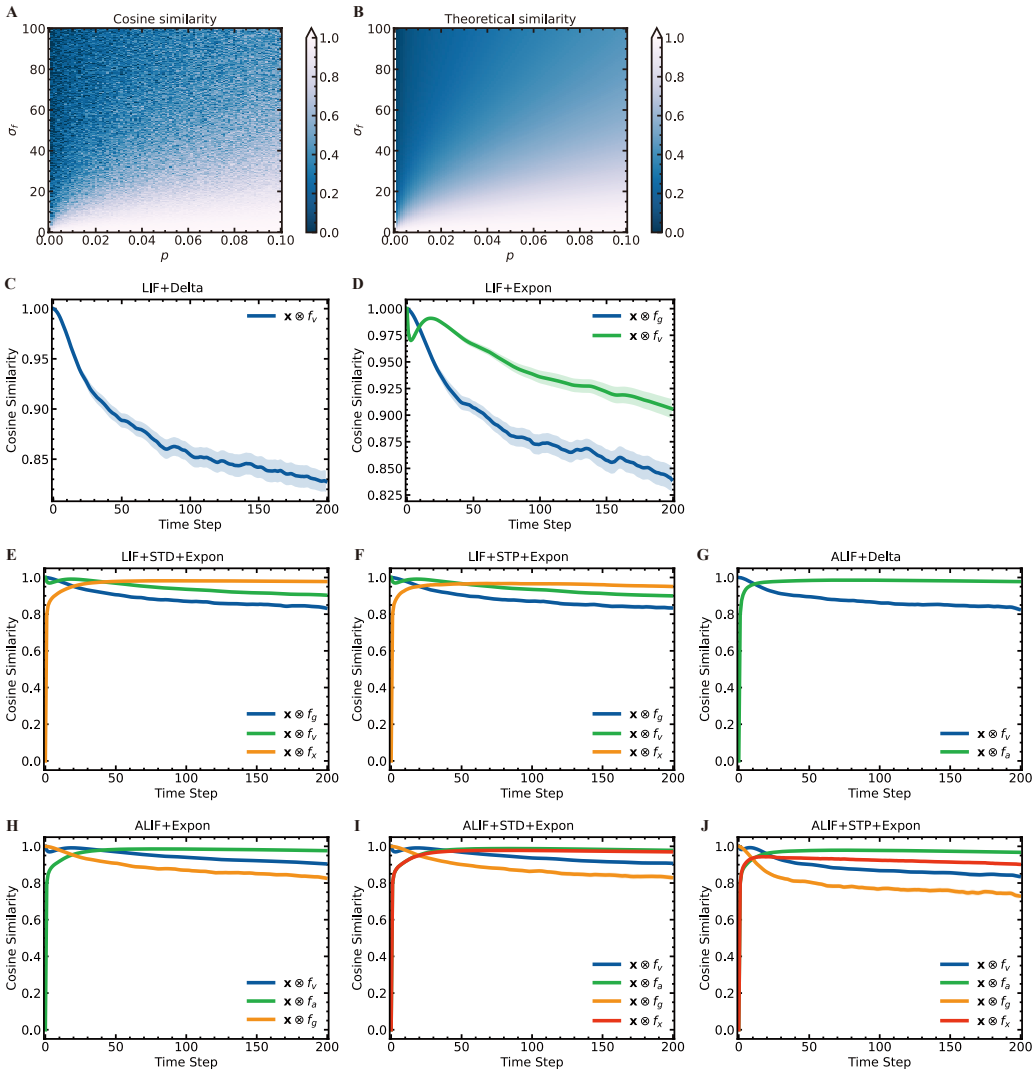

Figure S9: **Empirical Validation of Theorem 7.4 and Theorem 7.3**. (A) Phase portrait of cosine similarity as a function of presynaptic spiking probability p and the standard deviation of postsynaptic activation derivatives σ_f. The standard deviation σ_f varies widely from 1 to 100. The spiking probability p is varied between 0 and 0.1. Note: Higher spiking probabilities are omitted due to consistently high cosine similarity. (B) Theoretical cosine similarity calculated using Eq. 58, using the same parameter range as in (A). (C-J) Empirical evaluations of cosine similarity across various SNN dynamics while processing the IBM DVS Gestures dataset [1]. The x-axis represents the time step, and the y-axis denotes the cosine similarity. Each panel (C-J) corresponds to a different SNN architecture: (C) `LIF+Delta` dynamics (Eq. 43); (D) `LIF+Expon` dynamics (Eq. 45); (E) `LIF+STD+Expon` dynamics (Eq. 48); (F) `LIF+STP+Expon` dynamics (Eq. 50); (G) `ALIF+Delta` dynamics (Eq. 44); (H) `ALIF+Expon` dynamics (Eq. 46); (I) `ALIF+STD+Expon` dynamics (Eq. 49); (J) `ALIF+STP+Expon` dynamics (Eq. 51). Each line within a panel represents the cosine similarity between exact and approximated eligibility traces for the corresponding hidden states. For example, $\mathbf{x} \otimes f_{\mathbf{v}}$ in (C) shows the cosine similarity of the eligibility trace tracked by hidden state $\mathbf{v}$ in the `LIF+Delta` network.

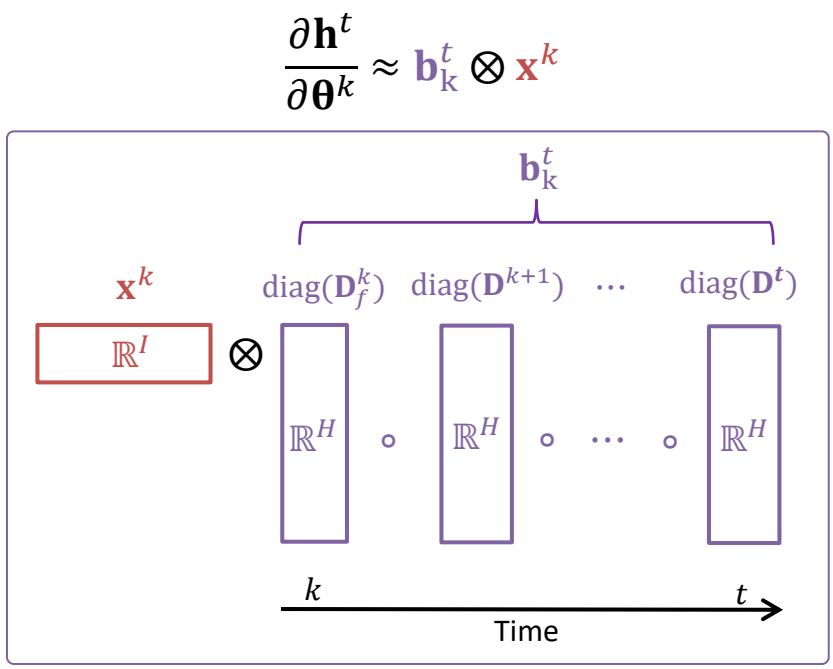

Figure S10: **Computation of the Eligibility trace $\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^k}$ in D-RTRL and ES-D-RTRL**. See the main text in Section 3.

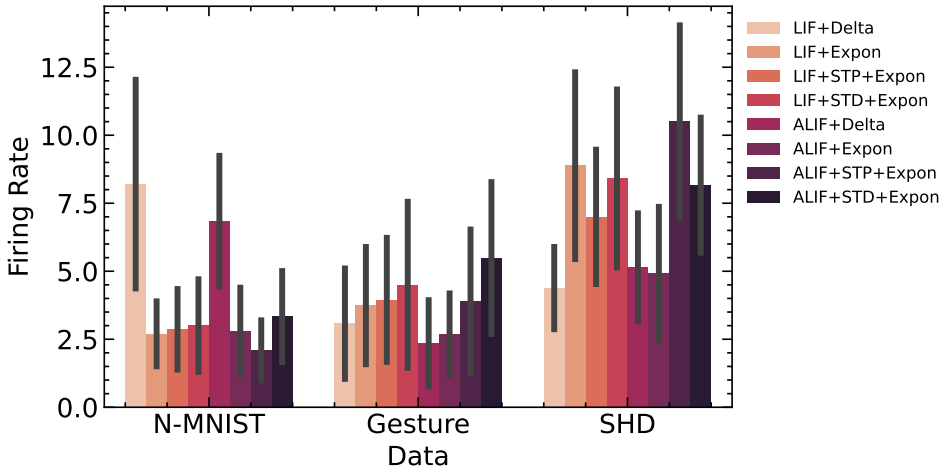

Figure S11: **Firing Rate Statistics During Jacobian and Gradient Evaluation**. This figure presents the firing rate statistics observed during the evaluation of Jacobian and gradient computations in Figure 5A-D across various network dynamics and neuromorphic datasets.

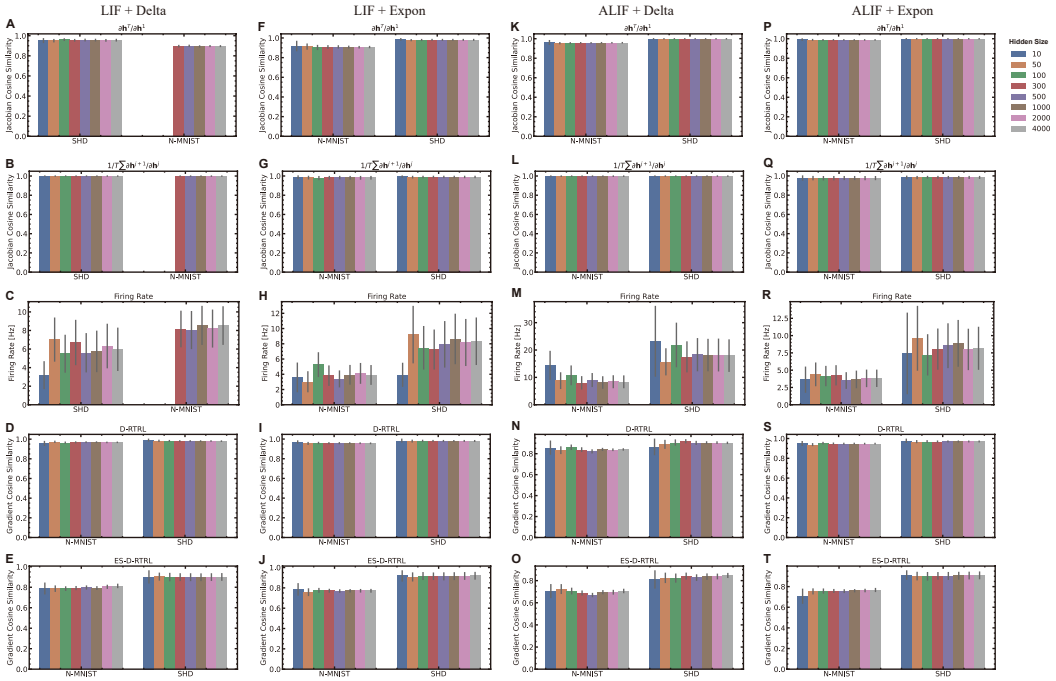

Figure S12: **Impact of Network Size on Cosine Similarity of Hidden Jacobian and Weight Gradients in Recurrent SNNs.** This figure presents a comprehensive analysis of how network size affects the cosine similarity of hidden Jacobian and weight gradients in various SNN configurations. Experiments were conducted using single recurrent layer architectures across different network sizes, utilizing the SHD [4] and N-MNIST [13] datasets. Each data point in the box plots represents the average of repeated experiments conducted on each dataset composed of 500 sampled data points, with 95% confidence intervals shown. The figure is organized into four columns and five rows, exploring different neuron dynamics and synapse models. *First Column (A-E)*: Evaluations by using LIF neuron dynamics connected with Delta synapse models. *Second Column (F-J)*: Evaluations by using LIF neuron dynamics connected with Exponential synapse models. *Third Column (K-O)*: Evaluations by using ALIF neuron dynamics connected with Delta synapse models. *Fourth Column (P-T)*: Evaluations by using ALIF neuron dynamics connected with Exponential synapse models. *First Row (A, F, K, P)*: The cosine similarity of long-term hidden Jacobian $\frac{\partial \mathbf{h}^T}{\partial \mathbf{h}^1}$. *Second Row (B, G, L, Q)*: The average cosine similarity of one-step hidden Jacobian $1/T \sum_{j=1}^{T} \frac{\partial \mathbf{h}^j}{\partial \mathbf{h}^{j-1}}$. *Third Row (C, H, M, R)*: The firing rate of the simulated single-layer SNN network. *Fourth Row (D, I, N, S)*: The cosine similarity of the weight gradients between BPTT and D-RTRL. *Fifth Row (E, J, O, T)*: The cosine similarity of the weight gradients between BPTT and ES-D-RTRL.

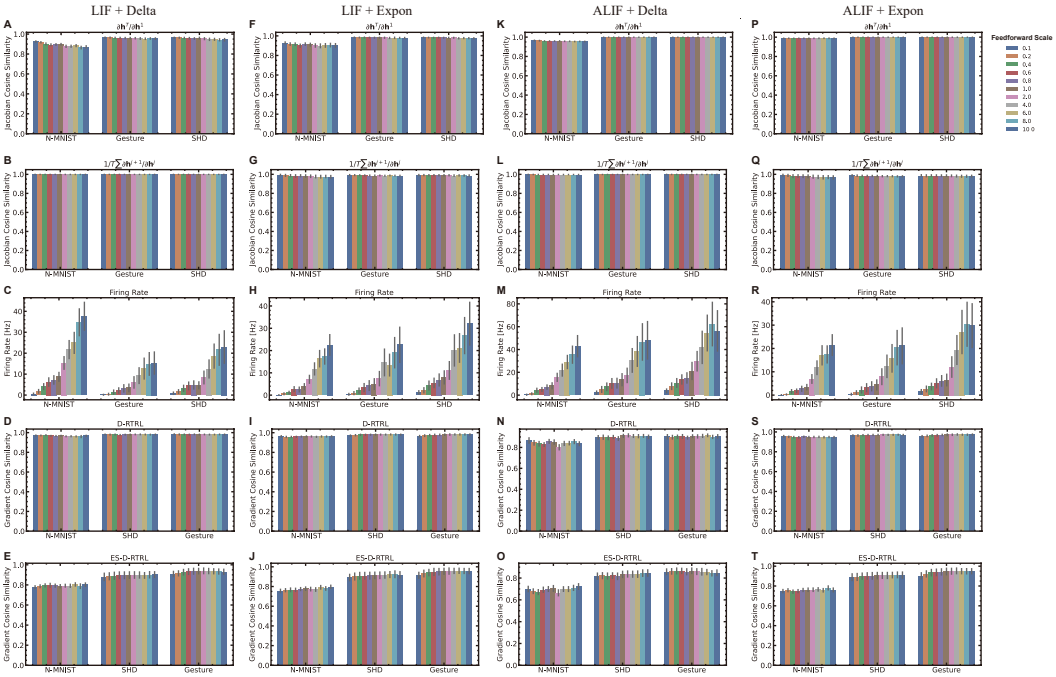

Figure S13: **Impact of Feedforward Connection Size on Cosine Similarity of Hidden Jacobian and Weight Gradients in Recurrent SNNs.** This figure presents a comprehensive analysis of how feedforward connection size affects the cosine similarity of hidden Jacobian and weight gradients in various SNN configurations. Experiments were conducted using single recurrent layer architectures across different feedforward connection scales, utilizing the N-MNIST [13], Gesture [1], and SHD [4] datasets. Each data point in the box plots represents the average of repeated experiments conducted on each dataset composed of 500 sampled data points, with 95% confidence intervals shown. The figure is organized into four columns and five rows, exploring different neuron dynamics and synapse models. *First Column (A-E)*: Evaluations by using LIF neuron dynamics connected with Delta synapse models. *Second Column (F-J)*: Evaluations by using LIF neuron dynamics connected with Exponential synapse models. *Third Column (K-O)*: Evaluations by using ALIF neuron dynamics connected with Delta synapse models. *Fourth Column (P-T)*: Evaluations by using ALIF neuron dynamics connected with Exponential synapse models. *First Row (A, F, K, P)*: The cosine similarity of long-term hidden Jacobian $\frac{\partial \mathbf{h}^T}{\partial \mathbf{h}^1}$. *Second Row (B, G, L, Q)*: The average cosine similarity of one-step hidden Jacobian $1/T \sum_{j=1}^{T} \frac{\partial \mathbf{h}^j}{\partial \mathbf{h}^{j-1}}$. *Third Row (C, H, M, R)*: The firing rate of the simulated single-layer SNN network. *Fourth Row (D, I, N, S)*: The cosine similarity of the weight gradients between BPTT and D-RTRL. *Fifth Row (E, J, O, T)*: The cosine similarity of the weight gradients between BPTT and ES-D-RTRL.

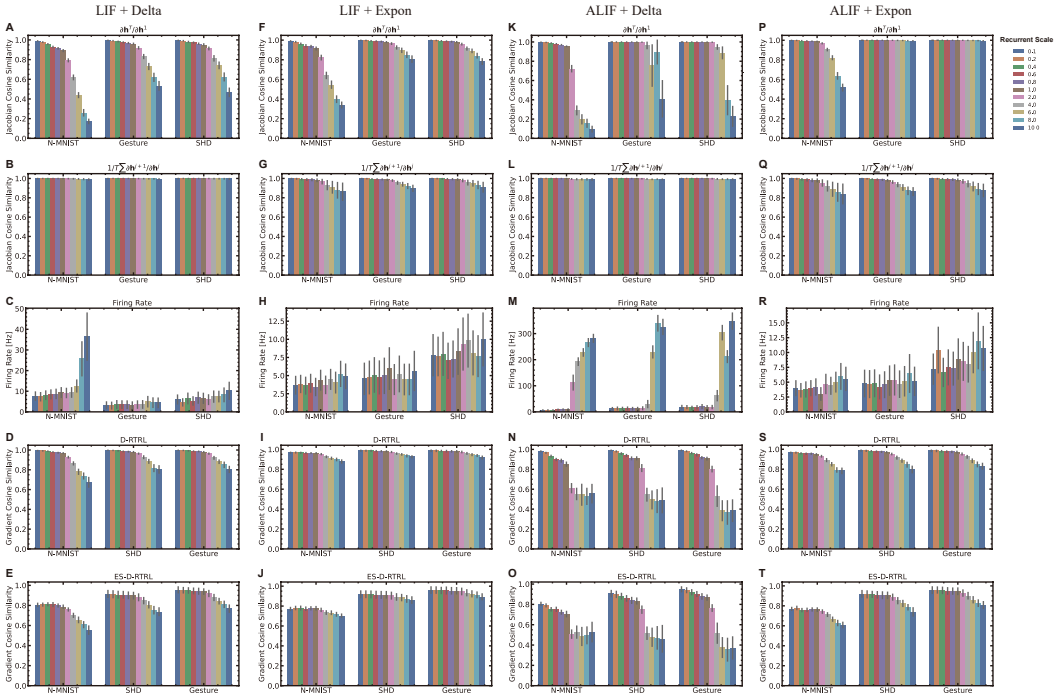

Figure S14: **Impact of Recurrent Connection Size on Cosine Similarity of Hidden Jacobian and Weight Gradients in Recurrent SNNs.** This figure presents a comprehensive analysis of how recurrent connection size affects the cosine similarity of hidden Jacobian and weight gradients in various SNN configurations. Experiments were conducted using single recurrent layer architectures across different recurrent connection scales, utilizing the N-MNIST [13], Gesture [1], and SHD [4] datasets. Each data point in the box plots represents the average of repeated experiments conducted on each dataset composed of 500 sampled data points, with 95% confidence intervals shown. The figure is organized into four columns and five rows, exploring different neuron dynamics and synapse models. *First Column (A-E)*: Evaluations by using LIF neuron dynamics connected with Delta synapse models. *Second Column (F-J)*: Evaluations by using LIF neuron dynamics connected with Exponential synapse models. *Third Column (K-O)*: Evaluations by using ALIF neuron dynamics connected with Delta synapse models. *Fourth Column (P-T)*: Evaluations by using ALIF neuron dynamics connected with Exponential synapse models. *First Row (A, F, K, P)*: The cosine similarity of long-term hidden Jacobian $\frac{\partial \mathbf{h}^T}{\partial \mathbf{h}^1}$. *Second Row (B, G, L, Q)*: The average cosine similarity of one-step hidden Jacobian $1/T \sum_{j=1}^{T} \frac{\partial \mathbf{h}^j}{\partial \mathbf{h}^{j-1}}$. *Third Row (C, H, M, R)*: The firing rate of the simulated single-layer SNN network. *Fourth Row (D, I, N, S)*: The cosine similarity of the weight gradients between BPTT and D-RTRL. *Fifth Row (E, J, O, T)*: The cosine similarity of the weight gradients between BPTT and ES-D-RTRL.

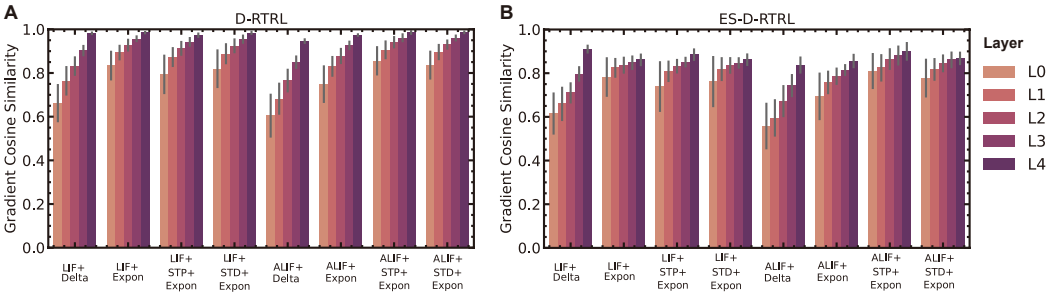

Figure S15: **Evaluation of Gradient Approximation Accuracy Over Different Layers**. we examined the impact of the depth of recurrent layers on the approximation of weight gradients in deep recurrent layers. We assessed the layer-wise cosine similarity between online gradient estimates and offline BPTT gradients using the SHD dataset [4]. Specifically, we evaluated a five-layer SNN network with 200 hidden units per layer, considering a variety of neuronal and synaptic dynamics (Methods 7.2). `L0` represents the bottom layer, and `L4` denotes the top layer. Consistently, we observed a decrease in cosine similarity as the layer depth increased (see panel A for the D-RTRL algorithm and panel B for the ES-D-RTRL algorithm). Lower layers exhibited lower cosine similarity to its their exact gradients, corresponding to the fact that more recurrence information of hidden states in top layers is ignored. (A) Layer-wise cosine similarity between weight gradients computed by D-RTRL and BPTT algorithms. (B) Layer-wise cosine similarity between weight gradients computed by ES-D-RTRL and BPTT algorithms.

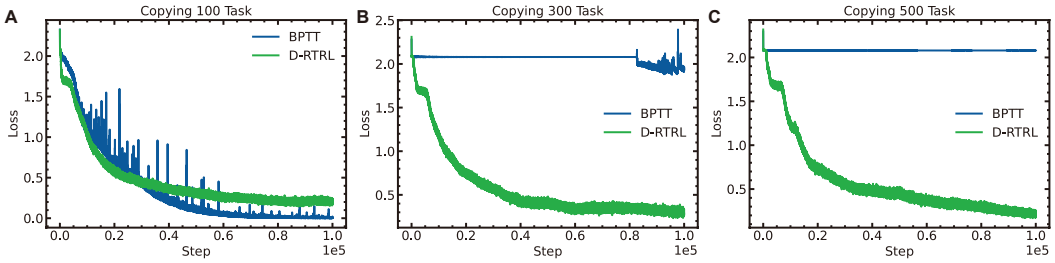

Figure S16: **Comparison of GRU Network Performance Using D-RTRL and BPTT Algorithms on the Copying Task**. This figure illustrates the cross-entropy loss curves for a GRU network trained using two different algorithms: D-RTRL and BPTT. The performance is evaluated on the copying task with varying sequence lengths. (A) The loss curve of copying task with sequence length $N = 100$. (B) The loss curve of copying task with sequence length $N = 200$. (C) The loss curve of copying task with sequence length $N = 300$. Each panel (A-C) shows: x-axis, training iterations or epochs; and y-axis, cross-entropy loss. This comparison allows for the assessment of the impact of increasing sequence length on algorithm performance, convergence rates and final loss values for each algorithm, and potential differences in learning stability between the two approaches.

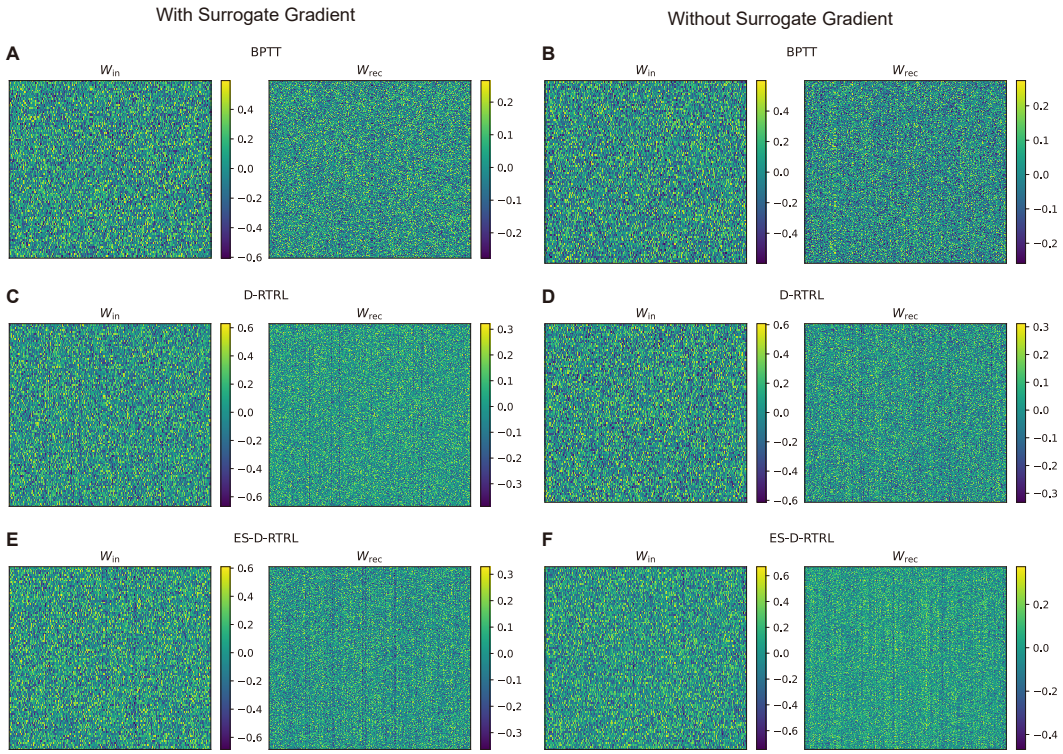

Figure S17: **Comparison of Weight Distributions in Spiking Networks Trained on the DMTS Task using Different Algorithms**. This figure presents weight visualizations for spiking networks trained on the DMTS task using three distinct algorithms: BPTT, D-RTRL, and ES-D-RTRL. The comparison is made both with and without the use of surrogate gradients. (A-B) Weights trained using BPTT. (C-D) Weights trained using D-RTRL. (E-F) Weights trained using ES-D-RTRL. Left column (A, C, E): Networks trained with surrogate gradients. Right column (B, D, F): Networks trained without surrogate gradients. Each panel shows a heatmap or color-coded representation of weight matrices. This visualization allows for a qualitative assessment of how different training algorithms and the use of surrogate gradients affect the final weight configurations in spiking neural networks trained on the DMTS task.

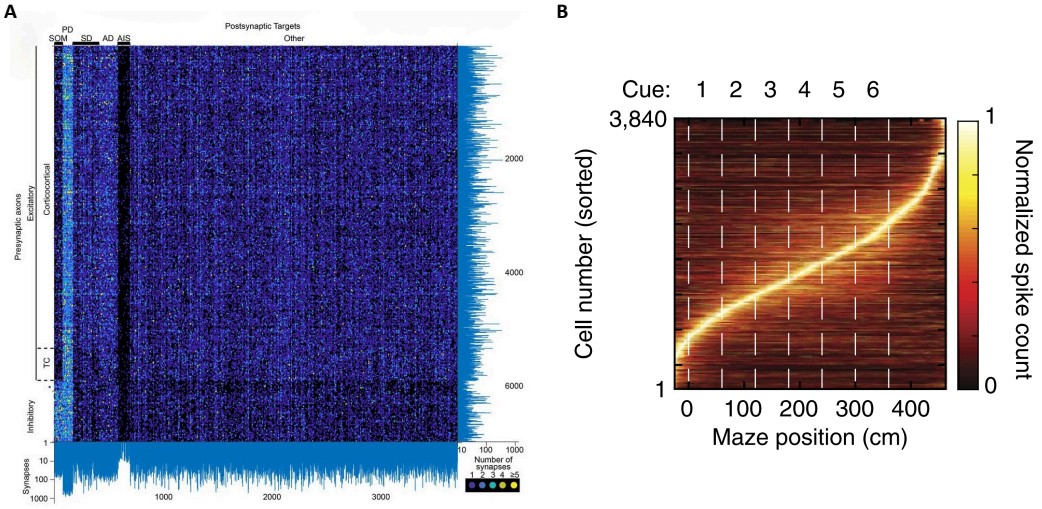

Figure S18: **Experimental Data Showing Synaptic Connectivity and Neural Activity Patterns Comparable to Our Simulation Results.**. (A) Layer 4 somatosensory cortical connectome between axons and postsynaptic targets [12]. This panel, adapted from Fig. 3E in [12], illustrates the connectome between all axons (n = 6,979) and postsynaptic targets (n = 3,719) within the volume, each with at least 10 synapses. A total of 153,171 synapses are established out of the 388,554 synapses detected in the volume. Please compare this panel with the data shown in Figure S17. (B) Sorted neuron activity in a head-restrained mouse performing a navigation-based evidence accumulation task [11]. This panel, adapted from Fig. 2a in [11], shows the normalized mean activity across all trials for all neurons in the posterior parietal cortex, pooled across all data sets (n = 3,840 cells from 5 mice). Traces were normalized to the peak of each cell's activity, averaged, and sorted by the peak's maze position. Please compare this panel with the data shown in Figure 6E-F.

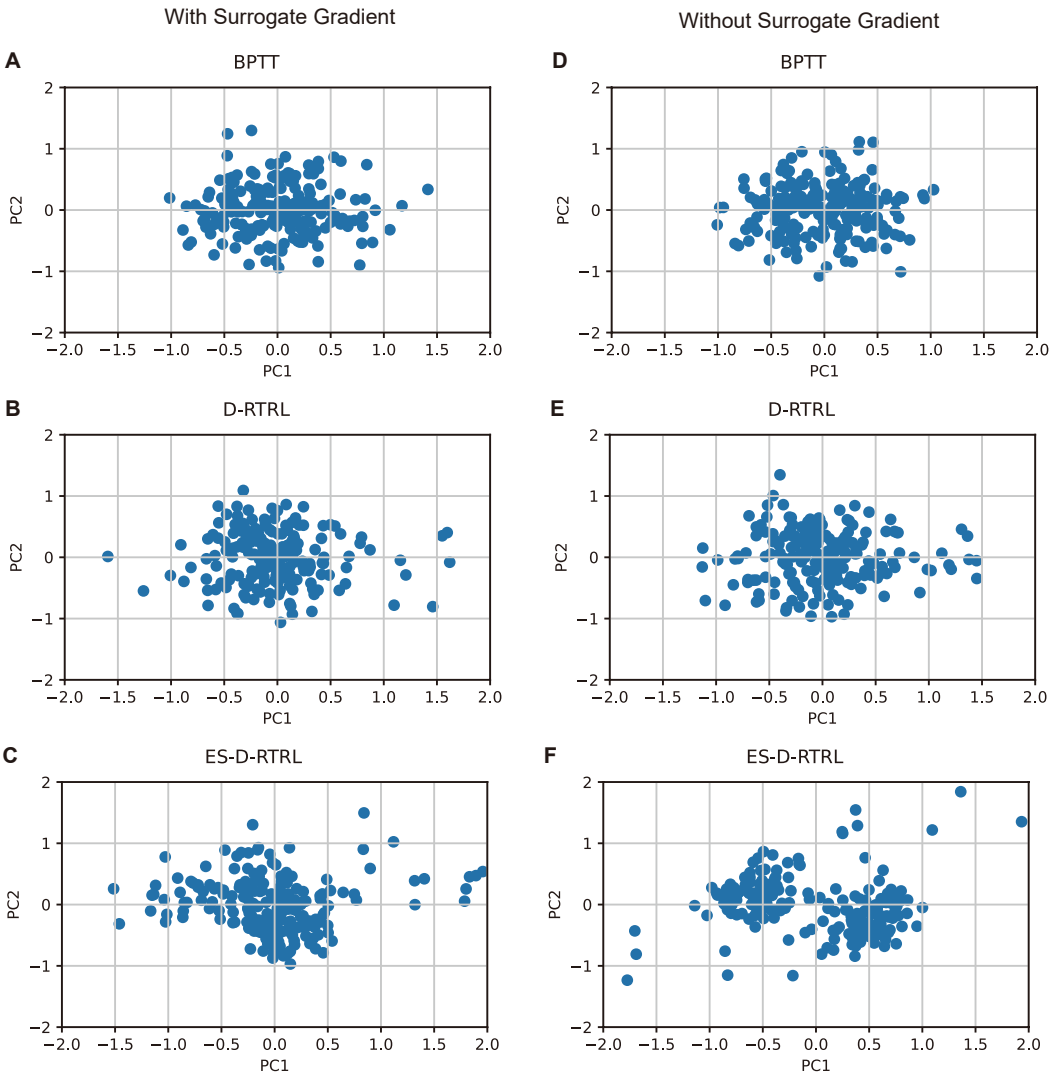

Figure S19: **Weight PCA visualization after trained by BPTT, D-RTRL, and ES-D-RTRL**. (A-B) Weights trained using BPTT. (C-D) Weights trained using D-RTRL. (E-F) Weights trained using ES-D-RTRL. Left column (A, C, E): Networks trained with surrogate gradients. Right column (B, D, F): Networks trained without surrogate gradients. Each panel shows a 2D scatter plot of weights projected onto the top 2 principal components.

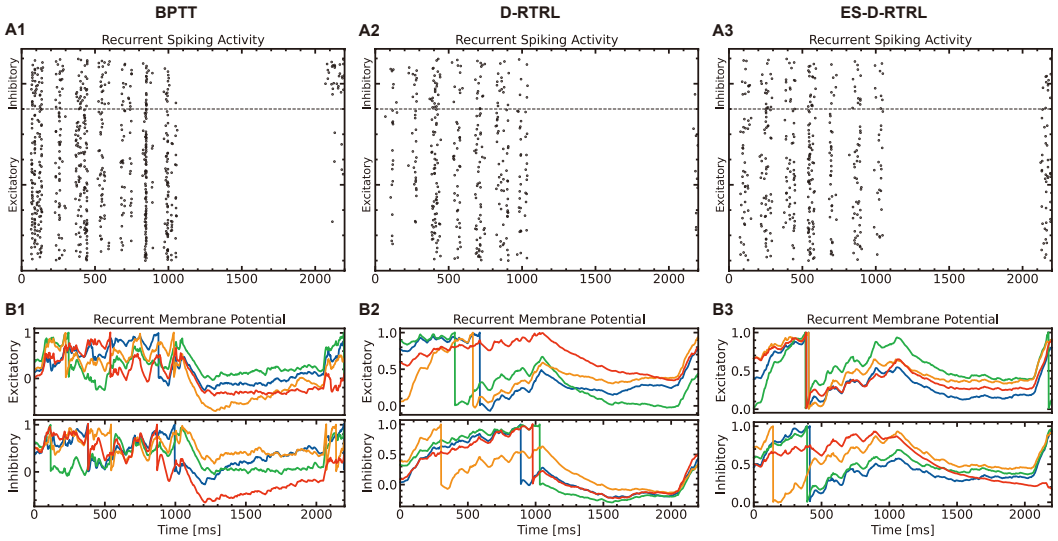

Figure S20: **Spiking and Membrane Potential Dynamics of the Network Trained on the Evidence Accumulation Task**. This figure shows the spiking and membrane potential dynamics of networks trained by BPTT, D-RTRL, and ES-D-RTRL on the evidence accumulation task (see Section 5.2 and Figure 6). (A1, B1) Recurrent dynamics of the network trained using BPTT. (A2, B2) Recurrent dynamics of the network trained using D-RTRL. (A3, B3) Recurrent dynamics of the network trained using ES-D-RTRL. (A) Spiking dynamics of the recurrent neurons. (B) Membrane potential dynamics of the recurrent neurons.

E Supplementary Tables

Parameter	Value
Learning rule	0.001
Learning optimizer	Adam [9]
Batch size	128
Time step Δt	1.0 ms

Table S2: Parameter settings for the learning hyper-parameters.

Parameter	Value
τ	10.0 ms
V_{v}	0.
V_{rest}	0.
V_{th}	1.
Hidden Size	single layer, 1000 neurons (Figure 3) single layer, 200 neurons (Figure 5 A-D) three layers, 512 neurons (Figure 5 I-L) three layers, 512 neurons (Table 1)
Input Size	1000 neurons (Figure 3) dataset dependent (Figure 5 A-D) dataset dependent (Figure 5 I-L) dataset dependent (Table 1)

Table S3: Parameter settings for the `LIF+Delta` network in Eq. 43. The corresponding network has been used in Figure 3, Figure 5, and Table 1.

Parameter	Value
Neuronal parameters	
τ	10.0 ms
V_{reset}	0.
V_{rest}	0.
V_{th}	1.
Synaptic parameters	
$\tau_{\mathbf{g}}$	10.0 ms
Hidden Size	single layer, 1000 neurons (Figure 3) single layer, 200 neurons (Figure 5 A-D)
Input Size	1000 neurons (Figure 3) dataset dependent (Figure 5 A-D)

Table S4: Parameter settings for the `LIF+Expon` network in Eq. 45. The corresponding network has been used in Figure 3 and Figure 5.

Parameter	Value
Neuronal parameters	
τ	10.0 ms
V_{reset}	0.
V_{rest}	0.
V_{th}	1.
Synaptic parameters	
$\tau_{\mathbf{g}}$	10.0 ms
τ_d	500.0 ms
Hidden Size	single layer, 1000 neurons (Figure 3) single layer, 200 neurons (Figure 5 A-D)
Input Size	1000 neurons (Figure 3) dataset dependent (Figure 5 A-D)

Table S5: Parameter settings for the `LIF+STD+Expon` network in Eq. 48. The corresponding network has been used in Figure 3 and Figure 5.

Parameter	Value
Neuronal parameters	
τ	10.0 ms
V_{reset}	0.
V_{rest}	0.
V_{th}	1.
Synaptic parameters	
$\tau_{\mathbf{g}}$	10.0 ms
τ_d	100.0 ms
τ_f	500.0 ms
Hidden Size	single layer, 1000 neurons (Figure 3) single layer, 200 neurons (Figure 5 A-D)
Input Size	1000 neurons (Figure 3) dataset dependent (Figure 5 A-D)

Table S6: Parameter settings for the `LIF+STP+Expon` network in Eq. 50. The corresponding network has been used in Figure 3 and Figure 5.

Parameter	Value
τ	10.0 ms
$\tau_{\mathbf{a}}$	100.0 ms
V_{reset}	0.0
V_{rest}	0.0
V_{th}	1.0
A	-0.1
Hidden Size	single layer, 1000 neurons (Figure 3) single layer, 200 neurons (Figure 5 A-D) three layers, 512 neurons (Figure 5 I-L) three layers, 200 neurons (Table 1)
Input Size	1000 neurons (Figure 3) dataset dependent (Figure 5 A-D) dataset dependent (Figure 5 I-L) dataset dependent (Table 1)

Table S7: Parameter settings for the `ALIF+Delta` network in Eq. 44. The corresponding network has been used in Figure 3, Figure 5, and Table 1.

Parameter	Value
Neuronal parameters	
τ	10.0 ms (Figure 3 and Figure 5 A-D)
τ	100.0 ms (Figure 5 E-H)
$\tau_{\mathbf{a}}$	100.0 ms (Figure 3 and Figure 5 A-D)
$\tau_{\mathbf{a}}$	1500.0 ms (Figure 5 E-H)
V_{reset}	0.
V_{rest}	0.
V_{th}	1.
A	-0.1 (Figure 3 and Figure 5 A-D)
A	-1.0 (Figure 5 E-H)
Synaptic parameters	
$\tau_{\mathbf{g}}$	10.0 ms (Figure 3 and Figure 5 A-D)
$\tau_{\mathbf{g}}$	100.0 ms (Figure 5 E-H)
Hidden Size	single layer, 1000 neurons (Figure 3) single layer, 200 neurons (Figure 5 A-D) single layer, 200 neurons (Figure 5 E-H)
Input Size	1000 neurons (Figure 3) dataset dependent (Figure 5 A-D) 100 neurons (Figure 5 E-H)

Table S8: Parameter settings for the `ALIF+Expon` network in Eq. 46. The corresponding network has been used in Figure 3 and Figure 5.

Parameter	Value
Neuronal parameters	
τ	10.0 ms
$\tau_{\mathbf{a}}$	100.0 ms
V_{reset}	0.
V_{rest}	0.
V_{th}	1.
A	-0.1
Synaptic parameters	
$\tau_{\mathbf{g}}$	10.0 ms
τ_d	500.0 ms
Hidden Size	single layer, 1000 neurons (Figure 3) single layer, 200 neurons (Figure 5 A-D)
Input Size	1000 neurons (Figure 3) dataset dependent (Figure 5 A-D)

Table S9: Parameter settings for the `ALIF+STD+Expon` network in Eq. 49. The corresponding network has been used in Figure 3 and Figure 5.

Parameter	Value
Neuronal parameters	
τ	10.0 ms
$\tau_{\mathbf{a}}$	100.0 ms
V_{reset}	0.
V_{rest}	0.
V_{th}	1.
A	-0.1
Synaptic parameters	
$\tau_{\mathbf{g}}$	10.0 ms
τ_d	100.0 ms
τ_f	500.0 ms
Hidden Size	single layer, 1000 neurons (Figure 3) single layer, 200 neurons (Figure 5 A-D)
Input Size	1000 neurons (Figure 3) dataset dependent (Figure 5 A-D)

Table S10: Parameter settings for the `ALIF+STP+Expon` network in Eq. 51. The corresponding network has been used in Figure 3 and Figure 5.

Parameter	Value
Neuronal parameters	
τ	100.0 ms
$\tau_{\mathbf{a}}$	1500.0 ms
V_{reset}	0.
V_{rest}	0.
V_{th}	1.
A	-1.0
Synaptic parameters	
τ_{e}	10.0 ms
τ_{i}	10.0 ms
E_{e}	5.0
E_{i}	-10.0
Network parameters	
Input Size	100 neurons
Hidden Size	single layer, 400 neurons
EI ratio	4:1
Recurrent connection probability	0.2

Table S11: Parameter settings for the E/I network in Eq. 52. The corresponding network has been used in Figure 6.

References

[1] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7243–7252, 2017.

[2] Minmin Chen. Minimalrnn: Toward more interpretable and trainable recurrent neural networks. arXiv preprint arXiv:1711.06788, 2017.

[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

[4] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. IEEE Transactions on Neural Networks and Learning Systems, 2020.

[5] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing, pages 6645–6649. Ieee, 2013.

[6] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752, 2023.

[7] Albert Gu, Caglar Gulcehre, Thomas Paine, Matt Hoffman, and Razvan Pascanu. Improving the gating mechanism of recurrent neural networks. In International conference on machine learning, pages 3800–3809. PMLR, 2020.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[10] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. arXiv preprint arXiv:1708.02182, 2017.

[11] Ari S Morcos and Christopher D Harvey. History-dependent variability in population dynamics during evidence accumulation in cortex. Nature neuroscience, 19(12):1672–1681, 2016.

[12] 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):eaay3134, 2019.

[13] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. Frontiers in neuroscience, 9:437, 2015.

[14] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In International Conference on Machine Learning, pages 26670–26698. PMLR, 2023.

[15] Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. arXiv preprint arXiv:2208.04933, 2022.

[16] Aaron Spieler, Nasim Rahaman, Georg Martius, Bernhard Schölkopf, and Anna Levina. The expressive leaky memory neuron: an efficient and expressive phenomenological neuron model can solve long-horizon tasks. In The Twelfth International Conference on Learning Representations, 2023.