nature communications

Article

https://doi.org/10.1038/s41467-026-68453-w

Model-agnostic linear-memory online learning in spiking neural networks

Received: 23 October 2024

Accepted: 5 January 2026

Published online: 19 January 2026

Check for updates

Chaoming Wang[1] ✉, Xingsi Dong[2,3], Zilong Ji[4], Mingqing Xiao[5], Jiedong Jiang[6], Xiao Liu[7], Yuxiang Huan[1] & Si Wu[1,2,3,8,9] ✉

Spiking neural networks (SNNs) offer a promising paradigm for modeling brain dynamics and developing neuromorphic intelligence, yet an online learning system capable of training rich spiking dynamics over long horizons with low memory footprints has been missing. Existing online approaches either incur quadratic memory growth, sacrifice biological fidelity through oversimplified models, or lack end-to-end automated tooling. Here, we introduce BrainTrace, a model-agnostic, linear-memory, and automated online learning system for spiking neural networks. BrainTrace standardizes model specification to encompass diverse neuronal and synaptic dynamics; implements a linear-memory online learning rule by exploiting intrinsic properties of spiking dynamics; and provides a compiler that automatically generates optimized online-learning code for arbitrary user-defined models. Across diverse dynamics and tasks, BrainTrace achieves strong learning performance with a low memory footprint and high computational throughput. Critically, these properties enable online fitting of a whole-brain-scale *Drosophila* SNN that recapitulates region-level functional activity. By reconciling generality, efficiency, and usability, BrainTrace establishes a foundation for spiking network modeling at scale.

Spiking neural networks (SNNs) offer a principled route to modeling neural computation at millisecond resolution while retaining the energy efficiency and temporal precision that characterize biological circuits[1,2]. Yet learning in SNNs remains fundamentally constrained by the challenge of long-horizon temporal credit assignment: synaptic updates must depend on state trajectories that unfold over thousands to millions of discrete time steps. The dominant solution, back-propagation through time (BPTT) with surrogate gradients[3–5], computes exact sequence-level gradients but does so offline, requiring storage of all intermediate states and thus incurring memory costs that scale linearly with sequence length T. For biological timescales, where one second may correspond to thousands of solver steps, this memory footprint quickly becomes prohibitive, limiting both the duration of trainable sequences and the size and complexity of SNN models[6].

Online learning algorithms for SNNs have emerged as a compelling alternative. By updating parameters in a forward-only fashion, they restrict memory usage to a small set of per-time-step variables and avoid storing entire trajectories[7]. However, despite encouraging progress, existing methods have not achieved both model-agnostic generality and linear memory scaling simultaneously. State-of-the-art approaches that approach BPTT-level accuracy, including e-prop[8] and OSTL[9], instantiate large collections of eligibility variables whose memory and computation scale quadratically with network size, rendering them impractical for large-scale spiking networks. Methods that

[1]Guangdong Institute of Intelligence Science and Technology, Hengqin, Zhuhai, Guangdong, China. [2]School of Psychological and Cognitive Sciences, Peking University, Beijing, China. [3]Peking-Tsinghua Center for Life Sciences, Academy for Advanced Interdisciplinary Studies, Peking University, Beijing, China. [4]Institute of Cognitive Neuroscience, University College London, London, England, UK. [5]Microsoft Research Asia, Shanghai, China. [6]Westlake Institute for Advanced Study, Westlake University, Hangzhou, Zhejiang, China. [7]Janelia Research Campus, Howard Hughes Medical Institute, Chevy Chase, MA, USA. [8]PKU-IDG/McGovern Institute for Brain Research, Peking University, Beijing, China. [9]Center of Quantitative Biology, Peking University, Beijing, China. ✉e-mail: wangchaoming@gdiist.cn; siwu@pku.edu.cn

achieve linear memory complexity[10–12] typically rely on restrictive neuron abstractions (e.g., simplified leaky integrate-and-fire (LIF) dynamics) that do not generalize to the rich neuronal dynamics needed for biological fidelity. Most critically for practice, there is no analogue of modern automatic differentiation (AD) ecosystems for online learning in SNNs: users cannot write arbitrary spiking dynamics and expect the system to synthesize numerically stable, efficient, and accurate online updates automatically. The absence of such an automated, model-agnostic framework has hampered adoption outside specialized groups and has limited real-world impact.

Here, we introduce BrainTrace, a model-agnostic, linear-memory, and automated online learning system for spiking neural networks that closes this gap. This system provides a theoretically guaranteed, linear-memory online gradient algorithm for training general spiking networks, together with a compiler that automatically generates efficient online-learning code from user-defined SNN models. For efficiency, BrainTrace leverages the sparse, event-driven nature of spiking neurons to reformulate real-time recurrent learning (RTRL)[13,14], which achieves linear memory scaling relative to the number of neurons while preserving gradient fidelity needed for effective training. For generality, this efficiency extends to a broad class of spiking network dynamics whose synaptic states can be organized along either presynaptic or postsynaptic axes. For usability, BrainTrace's compiler automatically generates efficient online learning code for any user-defined SNNs, facilitating implementations across various hardware platforms, including central processing units (CPUs), graphics processing units (GPUs), and tensor processing units (TPUs).

We validate BrainTrace across diverse SNN architectures and tasks. Relative to offline surrogate-gradient BPTT and online baselines with quadratic memory growth, BrainTrace delivers orders-of-magnitude lower memory consumption and high throughput, enabling long-horizon training on commodity accelerators. It attains strong training performance on standard neuromorphic classification benchmarks and brain-simulation settings, where it trains SNNs that reproduce neural dynamics underlying canonical cognitive tasks. Notably, BrainTrace scales to a whole-brain SNN of the *Drosophila* connectome[15,16] and successfully fits whole-brain functional activity patterns[17,18]. In sum, BrainTrace provides a unified solution to general, efficient, and automated SNN online learning, paving the way toward large-scale, biologically grounded spiking network modeling.

Results

BrainTrace supports a wide range of spiking networks

BrainTrace supports a wide range of SNNs with the following discretized dynamics (Supplementary Note A):

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

where $\mathbf{h}^t \in \mathbb{R}^{Hd}$ represents the hidden states of H neurons, with each neuron quantified by d state variables; $\mathbf{I}^t \in \mathbb{R}^H$ represents the synaptic currents; and t represents the time stamp; $\mathbf{x}^t \in \mathbb{R}^I$ represents synaptic inputs (e.g., presynaptic spikes); and $\boldsymbol{\theta} \in \mathbb{R}^P$ denotes connection weights between pre- and postsynaptic neurons (Fig. 1).

SNNs described by Eq. (1) are implemented using the `AlignPre` and `AlignPost` frameworks[19,20] (Fig. 1; see examples in Supplementary Note B). Specifically, in `AlignPre`, synapses with identical parameters that share the same presynaptic neuron exhibit identical dynamical trajectories (Supplementary Fig. 8), enabling synaptic variables to be aligned with the presynaptic neuronal dimension

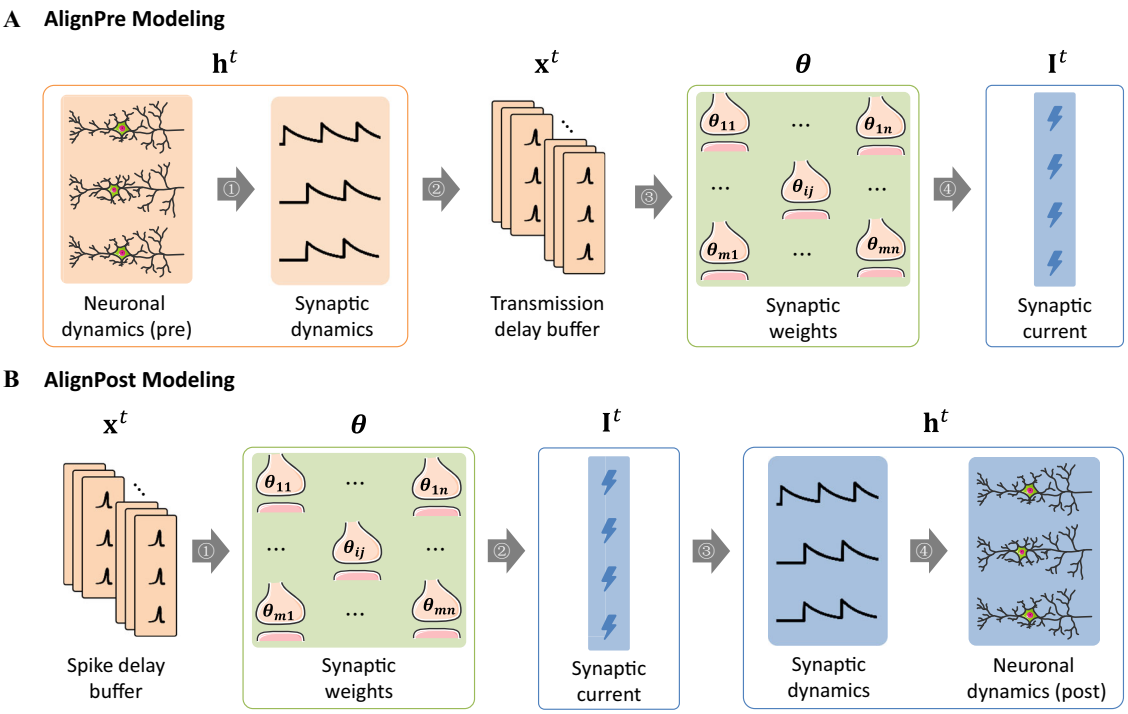


Fig. 1 | `AlignPre` and `AlignPost` abstractions for general spiking networks. **A** `AlignPre` modeling, where synaptic variables are aligned with the presynaptic neuronal dimensions. Presynaptic spikes drive the updates of synapses ①. The resulting delayed synaptic dynamics ② are then transmitted to postsynaptic sites ③, generating postsynaptic currents ④, which in turn influence the evolution of postsynaptic neuron dynamics. **B** `AlignPost` modeling, where synaptic variables are aligned with the postsynaptic neuronal dimensions. The delayed presynaptic spikes ① determine the postsynaptic inputs ②. These inputs drive the updates of synaptic dynamics ③, which then influence the evolution of postsynaptic neuron dynamics ④. In `AlignPre`, the input $\mathbf{x}$ governing neuronal interactions in Eq. (1) represents conductance of synaptic dynamics; while in `AlignPost`, the variable $\mathbf{x}$ in Eq. (1) corresponds to presynaptic spikes. The orange color represents the presynaptic dynamics with dimension $\mathbb{R}^m$, the blue color denotes the postsynaptic dynamics with dimension $\mathbb{R}^n$, and the green color represents the synaptic interaction $\boldsymbol{\theta}$ with dimension $\mathbb{R}^{m \times n}$.

(Fig. 1A). In `AlignPost`, when synapses feature exponential dynamics with identical time constants, multiple synapses from presynaptic neurons converging onto a single postsynaptic neuron can be collectively represented by a single shared exponential synapse. This shared representation captures the combined input to the postsynaptic neuron (Supplementary Fig. 8), allowing synaptic variables to be aligned along the postsynaptic dimension (Fig. 1B).

`AlignPre` and `AlignPost` decompose an SNN into two parts (Eq. (1)): (1) an "intrinsic dynamics" part that dictates the temporal evolution of neuronal states through the element-wise activation function f, and (2) a "neuronal interaction" part that converts presynaptic inputs into postsynaptic currents via the interaction operator $\circledast$. The decomposition can largely reduce computational complexity and memory consumption, enabling a simplified algorithm for SNN online learning (described in the next section).

BrainTrace achieves linear-memory online learning algorithm

The objective of SNN training is to find a group of synaptic weights $\boldsymbol{\theta}$ that minimizes a loss function $\mathcal{L}$ that quantifies the discrepancy between target values and network outputs. When target values are provided at discrete time points $t \in \mathcal{T}$, we compute the weight gradient by:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \sum_{t \in \mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \boldsymbol{\epsilon}^t, \tag{2}$$

where $\mathcal{L}^t$ represent the instantaneous loss at time t, $\partial \mathcal{L}^t / \partial \mathbf{h}^t$ denotes the learning signal that arrival at the hidden state $\mathbf{h}^t$, and $\boldsymbol{\epsilon}^t$ is the eligibility trace indicating how parameters influence the hidden state, which is computed as[13]:

$$\boldsymbol{\epsilon}^t = \frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} \boldsymbol{\epsilon}^{t-1} + \frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t}, \tag{3}$$

where the hidden Jacobian $\partial \mathbf{h}^t / \partial \mathbf{h}^{t-1} = \mathbf{D}^t + \mathbf{J}^t$ consists of a diagonal matrix $\mathbf{D}^t$ derived from intrinsic dynamics f, and a square matrix $\mathbf{J}^t$ derived from neuronal interactions $\circledast$ (see Methods). The hidden-to-weight Jacobian $\partial \mathbf{h}^t / \partial \boldsymbol{\theta}^t$ captures the effect of weight changes on hidden states. Importantly, Eq. (3) imposes a memory overhead of $\mathcal{O}(H^3)$ (Fig. 2A).

By leveraging SNN's inherent properties (see Methods), we can reduce the complexity to $\mathcal{O}(H)$. Without loss of generality, we consider

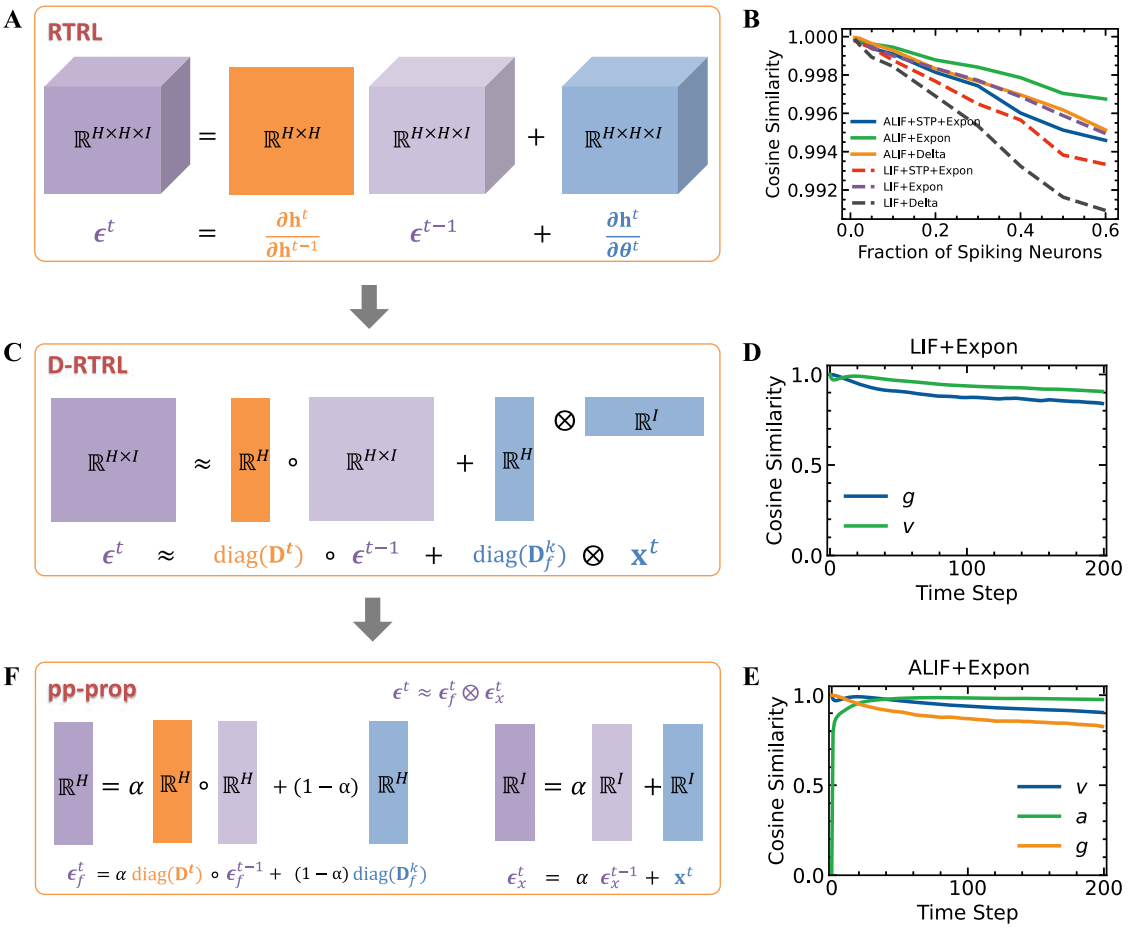


Fig. 2 | Deriving the linear-memory online learning algorithm. A The visualization of RTRL computation complexity reveals that $\boldsymbol{\epsilon}$ and the parameter Jacobian $\partial \mathbf{h}^t / \partial \boldsymbol{\theta}^t$ exhibit cubic memory complexity, while the hidden Jacobian $\partial \mathbf{h}^t / \partial \mathbf{h}^{t-1}$ demonstrates quadratic memory complexity. **B** Cosine similarity between the diagonal Jacobian $\mathbf{D}^t$ induced by intrinsic dynamics in Eq. (1) and the full hidden Jacobian $\partial \mathbf{h}^t / \partial \mathbf{h}^{t-1}$, over different network dynamics and fraction of spiking neurons. **C** In D-RTRL computation, the eligibility trace $\boldsymbol{\epsilon}$ is reduced to exhibit quadratic memory complexity. **D** Cosine similarity between the rank-one approximation (Eq. (5)) and D-RTRL eligibility trace (Eq. (4)) for LIF neurons with exponential synapse (Supplementary Eq. 72). Similarity is computed for membrane potential (v) and synaptic conductance (g). **E** Cosine similarity between the rank-one approximation (Eq. (5)) and D-RTRL eligibility trace (Eq. (4)) for ALIF neurons with exponential synapse (Supplementary Eq. 73). Similarity is computed for membrane potential (v), synapse (g), and adaptation (a) variables. **F** In pp-prop computation, the eligibility trace $\boldsymbol{\epsilon}$ is approximated using two variables with linear memory complexity that separately track pre-synaptic and post-synaptic neuronal activities.

each neuron has only 1 state variable with $d = 1$ (but see Methods for generalized results for $d > 1$). First, we found that a diagonal approximation of the hidden Jacobian captures over 99% of the cosine similarity with the full Jacobian ($\partial \mathbf{h}^t/\partial \mathbf{h}^{t-1} \approx \mathbf{D}^t$, Fig. 2B). This fact aligns with earlier practice[8], which suggests that ignoring the gradient flow through recurrent connections in ALIF neurons does not significantly hinder learning performance. Second, we proved (Supplementary Note C) that the hidden-to-weight Jacobian can be decomposed as $\partial \mathbf{h}^t/\partial \boldsymbol{\theta}^t = \mathbf{D}_f^t \otimes \mathbf{x}^t$, where $\mathbf{D}_f^t$ is a diagonal matrix with entries $\mathbf{D}_{f,ii}^t = \partial \mathbf{h}_i^t/\partial \mathbf{I}_i^t$, and $\otimes$ represents the Kronecker product. These observations enable us to simplify the eligibility trace in Eq. (3) to:

$$\boldsymbol{\epsilon}^t \approx \boldsymbol{\epsilon}_D^t = \mathbf{D}^t \boldsymbol{\epsilon}^{t-1} + \mathrm{diag}(\mathbf{D}_f^t) \otimes \mathbf{x}^t, \tag{4}$$

which we referred to as the diagonal-approximated RTRL (D-RTRL) (Fig. 2C). This diagonal approximation reduced the memory complexity from $\mathcal{O}(H^3)$ to $\mathcal{O}(H^2)$.

We further observed that $\boldsymbol{\epsilon}_D^t$ can be expressed as a summation of t rank-one matrices: $\boldsymbol{\epsilon}_D^t = \sum_{k=1}^t \mathbf{b}_k^t \otimes \mathbf{x}^k$, where $\mathbf{b}_k^t = \prod_{i=k+1}^t \mathrm{diag}(\mathbf{D}^i) \circ \mathrm{diag}(\mathbf{D}_f^k)$ (Supplementary Fig. 9). Since inputs $\mathbf{x}^t$ maintain consistent signs across time steps (whether as binary spikes in `AlignPost` or synaptic conductance in `AlignPre` (Fig. 1)), we proved that the sum of multiple rank-one matrices can be approximated as a single rank-one matrix between summed vectors (Supplementary Note D). Consequently, the eligibility trace can be simplified to:

$$\boldsymbol{\epsilon}_D^t \approx \frac{1}{t} \sum_{k=1}^t \mathbf{b}_k^t \otimes \sum_{k=1}^t \mathbf{x}^k, \tag{5}$$

which still preserves a high cosine similarity with the full eligibility trace across LIF (Fig. 2D), ALIF (Fig. 2E), and other SNN architectures (Supplementary Fig. 15).

In summary, this approximation yields the pre-post eligibility trace propagation (pp-prop) algorithm (Fig. 2F):

$$\boldsymbol{\epsilon}^t \approx \boldsymbol{\epsilon}_{\mathbf{f}}^t \otimes \boldsymbol{\epsilon}_{\mathbf{x}}^t, \tag{6}$$

$$\boldsymbol{\epsilon}_{\mathbf{x}}^t = \alpha \boldsymbol{\epsilon}_{\mathbf{x}}^{t-1} + \mathbf{x}^t, \tag{7}$$

$$\boldsymbol{\epsilon}_{\mathbf{f}}^t = \alpha\, \mathrm{diag}(\mathbf{D}^t) \circ \boldsymbol{\epsilon}_{\mathbf{f}}^{t-1} + (1-\alpha)\mathrm{diag}(\mathbf{D}_f^t), \tag{8}$$

where $\circ$ denotes element-wise multiplication, and α ($0 < \alpha < 1$) is the smoothing factor corresponding to time constant $\tau = -\Delta t/ln(\alpha)$. $\boldsymbol{\epsilon}_{\mathbf{x}} \in \mathbb{R}^I$ and $\boldsymbol{\epsilon}_{\mathbf{f}} \in \mathbb{R}^H$ represent eligibility traces tracking pre- and post-synaptic activity, respectively. The eligibility trace in pp-prop operates temporally, locally, and in an unsupervised way, effectively achieving memory complexity of $\mathcal{O}(H)$.

BrainTrace facilitates automated online learning compilation

The widespread success of ANNs primarily stems from the automation achieved in frameworks such as PyTorch[21] and TensorFlow[22] for their training algorithms, namely backpropagation and BPTT algorithms. To provide similar training convenience for SNNs, we introduced a compiler (Fig. 3A–E) that automates the translation of SNNs into efficient online learning implementations of D-RTRL and pp-prop, and subsequently deploys both models and generated algorithms across various hardware backends, including CPUs, GPUs, and TPUs (Fig. 3).

At the `Primitive` layer, the system encapsulates three primitives that capture the essential components of general SNNs (as formalized in Eq. (1), Fig. 3A). Specifically, `Hidden State` primitives represent internal neuronal and synaptic variables, such as membrane potentials and synaptic conductances; `Parameter` primitives denote trainable parameters, such as synaptic weights; while `Connection` primitives describe network connectivities, including dense, convolutional, sparse connections, and element-wise operations. By flexibly composing these primitives, the `Complex Network` layer supports the construction of complex SNNs with sophisticated dynamics and topologies (Fig. 3B).

The core innovation of the compiler lies in the `ETraceGraph` and `ETraceAlgorithm` layers. At the `ETraceGraph` stage, user-defined Python-based models are transformed into symbolic intermediate representations (IRs), which explicitly encode data dependencies between model parameters and neuronal states. The compiler analyzes IRs to extract higher-level computational graphs that describe how each model parameter, $\boldsymbol{\theta}$, influences the hidden state through specific connections $\circledast$. The resulting graph captures all critical information necessary for computing eligibility traces, including the hidden-to-hidden Jacobian ($\mathrm{diag}(\mathbf{D}^t)$) and hidden-to-weight Jacobians ($\mathrm{diag}(\mathbf{D}_f^t)$ and $\mathbf{x}^t$) (Fig. 3C).

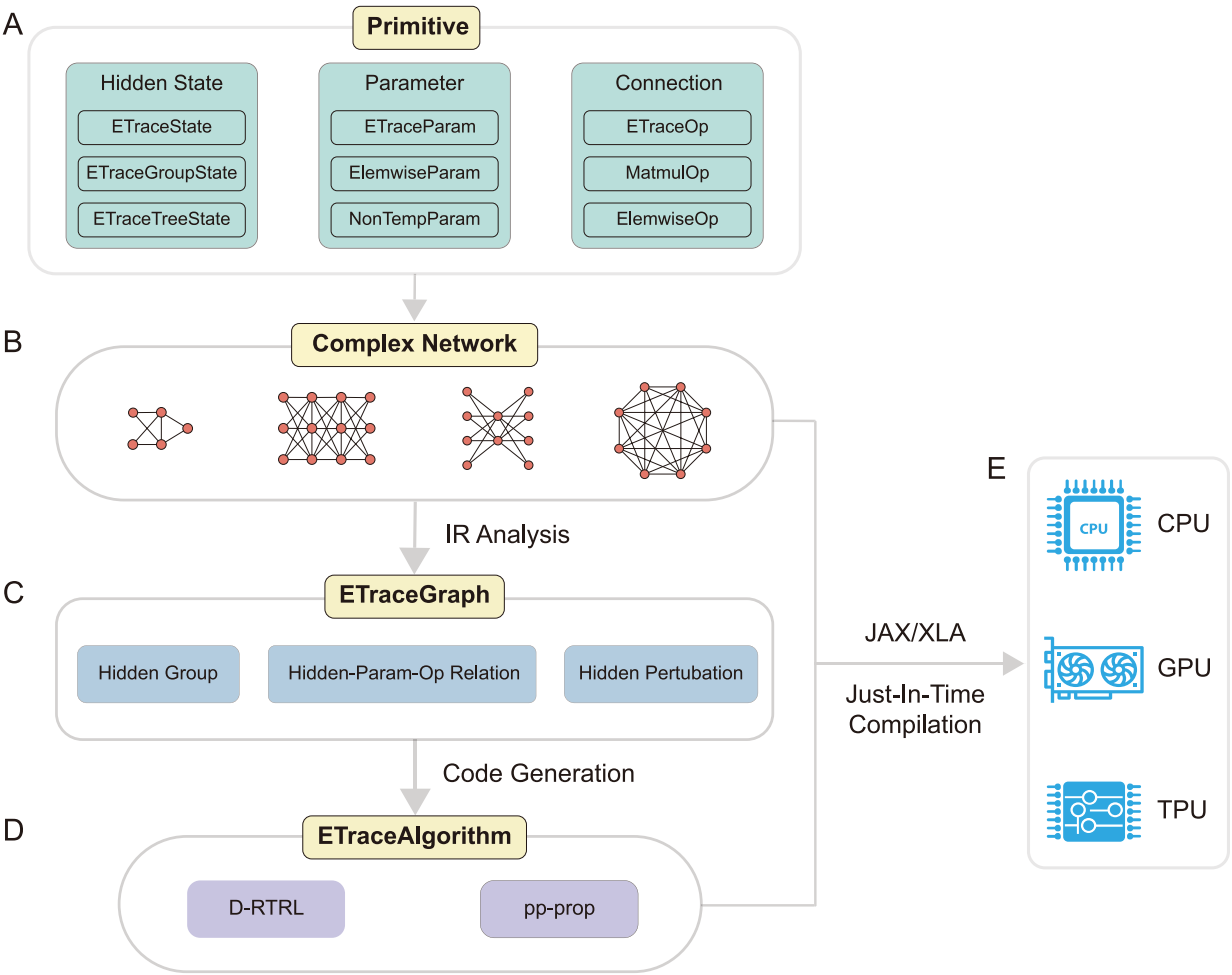
Subsequently, at the `ETraceAlgorithm` layer, the compiler automatically generates optimized online learning implementations based on the user-selected algorithm (Fig. 3D). The generated code includes both the logic for updating eligibility traces during state evolution (see Eq. (4) for D-RTRL and Eqs. (6)–(8) for pp-prop) and the gradient computation triggered by learning signals (see Eq. (2)). During this code generation process, the compiler also performs advanced optimizations, such as reusing shared eligibility traces within the same neuron population to minimize memory footprint, and vectorizing hidden-group Jacobian computations to maximize computational efficiency (Supplementary Note I).

Finally, leveraging just-in-time (JIT) compilation via JAX and XLA[23], both user-defined models (Fig. 3B) and compiler-generated algorithms (Fig. 3D) are automatically deployed onto the target hardware platform, including CPUs, GPUs, and TPUs (Fig. 3E).

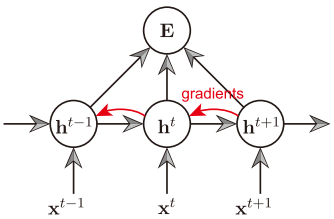
Through this hierarchical online learning compilation, BrainTrace provides a unified and automated solution for computing forward gradients (see comparison of Fig. 3F for BPTT and Fig. 3G for online learning) without introducing additional programming difficulties (see comparison of Fig. 3H for BPTT and Fig. 3I for online learning; Supplementary Note N). This end-to-end online learning system will greatly lower the barrier to entry for using online learning in SNNs, making advanced algorithms more accessible and deployable in real-world applications.

Algorithmic evaluations on approximation, long-term dependency, memory, and speed

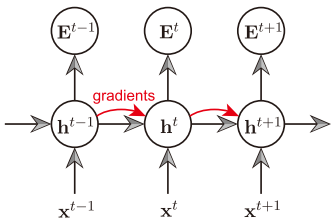
We first assessed the effectiveness of BrainTrace compiled algorithms in approximating the gradient computation. Evaluations were performed over different spiking datasets (including IBM DVS Gesture[24], Spiking Heidelberg Digits (SHD)[25], and Neuromorphic-MNIST (N-MNIST)[26] datasets), neuronal dynamics (ranging from LIF to ALIF neurons), and synapses (including Delta, exponential, short-term depression (STD), and short-term plasticity (STP)) (Supplementary Note B). We observed consistently high cosine similarity between the single-step hidden Jacobian ($1/T\sum_i^T \partial \mathbf{h}^{i+1}/\partial \mathbf{h}^i$; Fig. 4A) and long-range hidden Jacobian ($\partial \mathbf{h}^T/\partial \mathbf{h}^1$; Fig. 4B) compared to BPTT-computed Jacobian. Notably, high similarity values were especially prominent in networks with complex intrinsic dynamics, such as ALIF-based architectures (Fig. 4B). We further found that both D-RTRL and pp-prop achieved high gradient approximation accuracy relative to BPTT-computed gradients (Fig. 4C-D), validating their effectiveness for the gradient approximation. Moreover, both algorithms exhibited lower gradient similarities on N-MNIST (Fig. 4C-D), indicating some dataset dependence. Further investigation revealed several key insights about factors affecting the approximation accuracy (Supplementary Note G). Network size showed no significant impact on approximation accuracy (Supplementary Fig. 11). However, approximation declined greatly when either feedforward (Supplementary Fig. 12) or recurrent (Supplementary Fig. 13) weight



F BPTT gradient diagram



G Online learning gradient diagram



H BPTT pseudo code

```
01 import jax
02
03 def fn(sequences, targets):
04     losses = []
05     for seq, tar in zip(sequences, targets):
06         out = model(seq)
07         losses.append(loss_fn(out, tar))
08     return sum(losses)
09
10 gradients = jax.grad(fn)(sequences, targets)
```

I Online learning pseudo code

```
01 import jax, braintrace
02
03 def fn(seq, tar):
04     out = etmodel(seq)
05     return loss_fn(out, tar)
06
07 etmodel = braintrace.pp_prop(model)
08 gradients = ....  # zero initialization
09 for seq, tar in zip(sequences, targets):
10     gradients += jax.grad(fn)(seq, tar)
```

strength increased to levels that produced unrealistically high firing rates. Additionally, deeper recurrent layers led to lower approximation scores (Supplementary Fig. 14), suggesting a preference for shallow network architectures in our algorithms.

We then assessed whether BrainTrace algorithms can capture long-term dependencies in datasets, a capability that is critical for effective SNN modeling. We trained recurrent spiking networks (Supplementary Eq. 73) on the delayed match-to-sample (DMTS) task[27], in which the network is required to maintain directional information from the sample phase over a long delay period before making a decision based on the test period stimulus (see Methods). We found that while D-RTRL, pp-prop, and BPTT all effectively solve the task with

Fig. 3 | The architecture of online learning compilation. A Three types of primitives are defined to structure spiking network modeling: `Hidden State` for neuronal and synaptic states, `Parameter` for trainable parameters, and `Connection` for synaptic connectivity types. **B** Three primitives enable the construction of spiking networks with any complex network dynamics and topologies. **C** BrainTrace uses IR analysis to abstract Python-defined models into symbolic representations, then analyzes the graphs between hidden groups, weight parameters, and synaptic connections. **D** Based on the analyzed graphs, the compiler generated code implementing the user-selected algorithm-either D-RTRL or pp-prop. **E** Using the JIT compilation capability of JAX and XLA, both user-coded models and compiler-generated algorithms are compiled and executed on CPUs, GPUs, and TPUs. **F** Gradient computation diagram for BPTT autograd implementation, showing backward gradient accumulation. **G** Gradient computation diagram for online D-RTRL and pp-prop implementation, showing forward gradient accumulation. **H** Pseudo code for the implementation of BPTT gradient diagram, where the `jax.grad` function processes the complete sequence at once. **I** Pseudo code for the implementation of online learning gradient diagram, where `jax.grad` is applied at each step, and gradient computation is performed in forward mode as the sequence unrolls temporally.

surrogate gradients[3] (Fig. 4E), only D-RTRL and pp-prop can learn successfully without surrogate gradients (Fig. 4F). We then investigated how our algorithm achieves long-term dependency learning. We found that both pp-prop and D-RTRL maintain temporal information through eligibility traces, but in distinct ways. In D-RTRL, $\boldsymbol{\epsilon}_{\mathbf{g}}$ responds to incoming spikes, increasing with each presynaptic event; $\boldsymbol{\epsilon}_{\mathbf{a}}$ captures slow Jacobian dynamics, enabling information to persist across delays; and $\boldsymbol{\epsilon}_{\mathbf{v}}$ combines information from $\boldsymbol{\epsilon}_{\mathbf{g}}$ and $\boldsymbol{\epsilon}_{\mathbf{a}}$, remaining sensitive to input changes like $\boldsymbol{\epsilon}_{\mathbf{g}}$ while decaying gradually like $\boldsymbol{\epsilon}_{\mathbf{a}}$ (Fig. 4G). In pp-prop, the presynaptic trace $\boldsymbol{\epsilon}_{\text{pre}}$ is driven solely by presynaptic spikes, while $\boldsymbol{\epsilon}_{\text{post},\mathbf{g}}$ tracks postsynaptic currents and remains constant; $\boldsymbol{\epsilon}_{\text{post},\mathbf{a}}$ and $\boldsymbol{\epsilon}_{\text{post},\mathbf{v}}$ evolve slowly, propagating long-term postsynaptic Jacobian information until learning signals arrive during testing (Fig. 4H). We also analyzed the differences in synaptic connectivity learned by pp-prop compared to BPTT and D-RTRL. pp-prop maintained similar weight distributions with D-RTRL and BPTT, yet it displayed distinct column structures, particularly in recurrent weights (Supplementary Fig. 21). Interestingly, similar column structures have been observed in the synaptic connectome of layer 4 in the somatosensory cortex (Fig. 3E in[28]), suggesting that pp-prop may capture biologically relevant connectivity patterns.

Finally, we assessed the memory consumption and computational speed of pp-prop compared with other algorithms. Theoretically, for memory consumption, pp-prop requires only $\mathcal{O}(BN)$ memory, since it stores just two traces capturing pre- and postsynaptic activities, whereas other online learning methods demand at least $\mathcal{O}(BN^2)$. Here, B is the batch size. For computation, pp-prop needs $\mathcal{O}(TBN^2)$ operations for gradient updates but only $\mathcal{O}(TBN)$ operations for eligibility traces. In contrast, OSTL, e-prop, and D-RTRL incur quadratic costs for eligibility trace computation due to their reliance on matrix-based updates. Empirically, we ran experiments on the Gesture dataset to compare memory and speed of three training methods, BPTT, D-RTRL, and pp-prop, across different sequence lengths using LIF networks (Supplementary Eq. 70). We found that BPTT's GPU memory usage grew linearly with time steps and crashed at 400 steps (Fig. 4J). By contrast, both D-RTRL and pp-prop required constant memory regardless of sequence length (Fig. 4J), with pp-prop being 35 times more memory-efficient than D-RTRL (0.5 GB vs. 17.5 GB). For computation time, all algorithms scaled linearly with time steps, but pp-prop ran slightly faster than BPTT and achieved a tenfold speedup over D-RTRL (Fig. 4K). These memory and speed advantages were consistent on CPU (Supplementary Fig. 19) and TPU platforms (Supplementary Fig. 20). Additionally, we tracked model accuracy during training and found that all three algorithms achieved higher test accuracy as sequence length increased (Fig. 4L).

Performance evaluations on neuromorphic classification tasks

Spiking networks are widely applied in neuromorphic settings[1,6]. To assess the effectiveness of pp-prop for such applications, we conducted extensive experiments across three standard benchmarks: N-MNIST[26], SHD[25], and IBM DVS Gesture[24]. We evaluated multiple SNN architectures: (i) a two-layer RLIF network[29] comprising LIF neurons with delta synapses and recurrent connections (Supplementary Eq. 70); (ii) a two-layer RadLIF network[29] with adaptive LIF neurons, delta synapses, and recurrence (Supplementary Eq. 71); and (iii) a one-layer event-based gated recurrent unit (EGRU)[30] (Supplementary Eq. 82).

Since performance depends on many confounders (such as data preprocessing, network architecture, and training schedule), we varied only the learning algorithm while keeping all other factors fixed. For each algorithm, we performed targeted hyperparameter searches to obtain strong settings (see Supplementary Note L). We report mean ± s.d. accuracy over at least three independent runs (Table 1). Across datasets and architectures, pp-prop matched BPTT on most settings while using an online, memory-efficient update rule.

Against state-of-the-art online learners, including OSTL[9], e-prop[8], ETLP[31], OSTTP[32], OTPE[11], S-TLLR[33], FPTT[34], and OTTT[10], pp-prop consistently achieved the strong performance. Notably, on SHD, pp-prop exceeded the strongest baseline by more than 10 percentage points (Table 1).

Compared with state-of-the-art offline learning approaches, our pp-prop algorithm achieves comparable performance to the latest models that employ advanced architectural and training optimizations. For instance, on the SHD dataset, pp-prop even surpasses the best-performing models incorporating delay-based learning[35]. On the DVS Gesture benchmark, its performance closely matches that of models leveraging batch normalization[36] and specialized neural dynamics design[37].

Training excitatory-inhibitory spiking networks for cognitive tasks

BrainTrace is theoretically applicable to a broad spectrum of spiking networks, including biologically realistic circuits with complex neuronal dynamics. To demonstrate this, we developed a biologically-informed excitatory-inhibitory spiking network (Fig. 5B) incorporating: four-variable generalized leaky integrate-and-fire (GIF) neurons[38] for complex neuronal dynamics, distinct excitatory and inhibitory (E/I) populations, conductance-based synapses[39], and sparse connectivity[40]. Our network comprised 800 GIF neurons, with a 4:1 excitatory-to-inhibitory ratio and a 10% random connection probability, each featuring conductance-based exponential synapse dynamics (Supplementary Eq. 81). We trained this network on an evidence-accumulation task[41], where animals navigate a virtual linear track and encounter seven visual cues (Fig. 5A). The task required animals to integrate left/right visual cues and, upon reaching a T-junction, select the corresponding direction. Correct decisions based on majority cue distribution were rewarded.

Our analysis revealed that when the readout layer decodes recurrent membrane potential dynamics for accuracy, both D-RTRL and e-prop achieved stable convergence on this task, whereas BPTT failed without surrogate gradients (Fig. 5C, Supplementary Fig. 25). Despite the network's small size, BPTT required over 10 GB of device memory, and D-RTRL consumed more than 2 GB. In contrast, pp-prop maintained high performance with a memory footprint of less than 0.2 GB (Fig. 5D). Post-training analysis of pp-prop network dynamics revealed highly selective, temporally precise spiking patterns. During the evidence accumulation phase (0 - 1100 ms, Fig. 5E),

https://doi.org/10.1038/s41467-026-68453-w

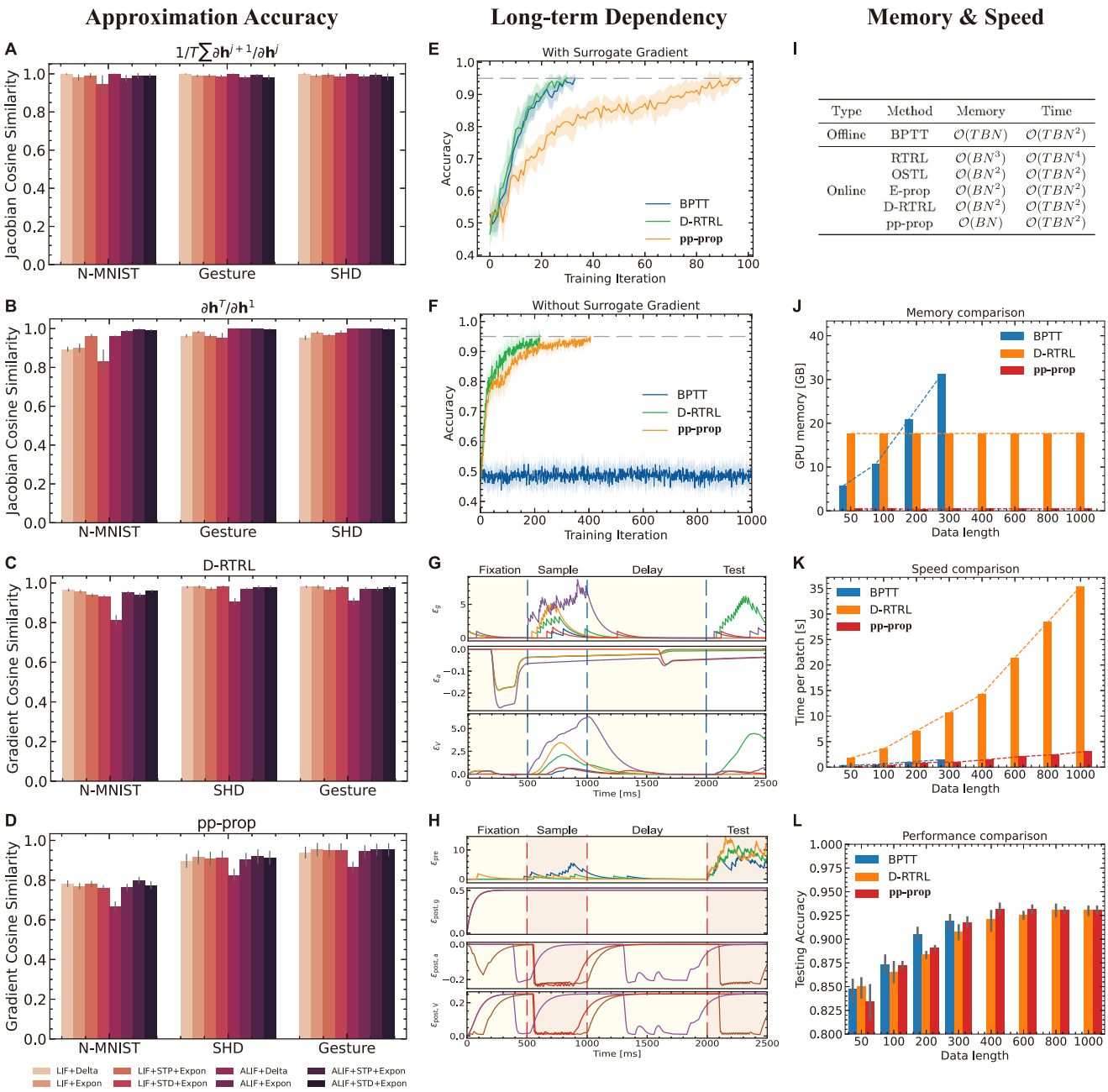


Fig. 4 | Evaluation of learning algorithms on approximation accuracy, long-term dependency learning, memory usage, and computational speed. **A**–**D** Approximation accuracy evaluations across three neuromorphic datasets and eight network dynamics. For each dataset, we randomly selected 500 data points, 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. **B** Cosine similarity between approximated and exact long-range hidden Jacobians. **C** Cosine similarity between weight gradients computed by D-RTRL and BPTT algorithms. **D** Cosine similarity between weight gradients computed by pp-prop 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. D-RTRL maintains three eligibility traces: $\boldsymbol{\epsilon}_{\mathbf{g}}$, $\boldsymbol{\epsilon}_{\mathbf{a}}$, and $\boldsymbol{\epsilon}_{\mathbf{v}}$, corresponding to the hidden states **g**, **a**, and **v**. **H** Visualization of the eligibility trace in the pp-prop 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[24]. **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, with results averaged over ten runs and reported with 95% confidence intervals.

sequentially presented visual cues induced orderly transitions between transient and clustered neuron population activities (Fig. 5F). 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 (Fig. 5F). Remarkably, these neural dynamics closely align with neuronal activities observed in the posterior parietal cortex of mice performing similar evidence accumulation tasks (Fig. 2a in ref. 41).

Table 1 | Comparison of training methods and network architectures across different datasets

Dataset	Category	Training method	Network architecture	Testing accuracy
N-MNIST[26]	Ours	BPTT	RLIF	98.33 ± 0.04%
		pp-prop	RLIF	98.25 ± 0.03%
		BPTT	RadLIF	98.29 ± 0.02%
		pp-prop	RadLIF	**98.40** ± 0.03%
	Online	ETLP[31]	RLIF	94.30%
		e-prop[31]	RLIF	97.90%
		OSTL[9]	sSNU	96.80 ± 0.17%
SHD[25]	Ours	BPTT	RLIF	94.01 ± 0.12%
		pp-prop	RLIF	93.93 ± 0.28%
		BPTT	RadLIF	94.72 ± 1.06%
		pp-prop	RadLIF	**95.33** ± 0.11%
	Online	ETLP[31]	RLIF	78.71 ± 1.49%
		e-prop[31]	RLIF	80.79 ± 0.39%
		OTPE[11]	LIF	76.70 ± 0.70%
		e-prop[60]	TC-RLIF	80.57%
		OSTTP[32]	SNU	77.33 ± 0.8%
		S-TLLR[33]	RLIF	78.24 ± 1.84%
	Offline	EventProp[61]	RLIF	93.50 ± 0.70%
		BPTT[35]	DCLS-Delays	95.07 ± 0.24%
		BPTT[29]	RadLIF	94.62%
Gesture[24]	Ours	BPTT	RLIF	93.97 ± 0.15%
		pp-prop	RLIF	94.26 ± 0.32%
		BPTT	EGRU	97.45 ± 0.27%
		pp-prop	EGRU	97.29 ± 0.16%
	Online	FPTT[34]	RLIF	92.13 ± 0.87%
		FPTT[34]	CNN	97.22%
		OTTT[10]	VGG-11	96.88%
	Offline	BPTT[62]	STS-ResNet	96.7%
		BPTT[63]	STL-SNN	97.01 ± 0.23%
		BPTT[37]	PLIF	**97.57%**
		BPTT[36]	VGGSNN	**97.57%**

Results show testing accuracy with standard deviation for various training approaches, including BPTT and pp-prop applied to different network architectures (RLIF, RadLIF) on N-MNIST, SHD, and Gesture datasets. State-of-the-art online and offline learning methods are included for comparison. The bolded data signifies the optimal performance.

Modeling *Drosophila* resting-state functional connectivity via whole-brain spiking network training

The linear memory complexity of BrainTrace provides a crucial advantage for training large-scale spiking networks. To illustrate this, we trained a whole-brain spiking model of the *Drosophila* brain constrained by the FlyWire anatomical connectome[15], to reproduce both resting-state whole-brain calcium imaging signals and their inter-regional functional connectivity[17,18]. Although the FlyWire connectome[15] offers a detailed anatomical map of neuronal wiring, direct implementation of this connectivity in spiking network models[16] fails to accurately capture either the resting-state neural activity in each neuropil (Fig. 6G) or the cross-neuropil correlations (Fig. 6K) recorded in experiments. We therefore trained the synaptic weights of the connectome-based network[16] using our pp-prop algorithm.

We first converted the 17 minutes of calcium signals recorded from each brain region (Fig. 6A) into deconvolved firing rates (Fig. 6B), and divided the entire dataset into a training set of 80% and a testing set of 20%. During training, the network received the experimental deconvolved firing rate at time t (ExpFR^t) as input and predicted the firing rate at time $t+1$ (SimFR^{t+1}). The loss function was defined as the mean squared error between simulated and experimental firing rates (Fig. 6C). The pp-prop algorithm then continuously adjusted synaptic weights online according to the error gradients, enabling the network to autonomously learn the complex spatiotemporal patterns underlying resting state dynamics. This model consumes approximately 8.9 GB of GPU memory with pp-prop, and the loss converges slowly over epochs (Fig. 6D). In contrast, training with D-RTRL and BPTT exceeds the 32 GB capacity of a single GPU (Supplementary Fig. 24).

At the neuropil level, we evaluated the model's dynamics in the mushroom body to assess training-induced improvements (Fig. 6E–G). In the warmup phase, we initialized whole-brain dynamics by feeding the first 84 seconds of experimental firing rates into the model; thereafter, without external inputs, the model generated ongoing neural activity solely from its previous timestep's neuropil state. Before training, the model failed to reproduce any experimentally observed spontaneous fluctuations (Fig. 6G); after training, however, its output not only exhibited activation patterns highly consistent with the experimental recordings ("Train" phase in Fig. 6F) but also sustained similar oscillation patterns on previously novel test data ("Test" phase in Fig. 6F). These results demonstrate the model's ability to generalize to untrained neural dynamics. To quantify improvements in all brain regions across the entire *Drosophila* brain, we computed Pearson correlations between model-generated and experimental firing rates for each of 68 regions, both pre- and post-training. Training produced a marked uplift in regional fit, and statistical tests confirmed that nearly every region exhibited a significant post-training correlation increase (Fig. 6H).

At the whole-brain level, we further compared whole-brain functional connectivity among experimental recordings, the untrained model's spontaneous activity, and the trained model's output (Fig. 6I–K). Prior to training, neurons within each neuropil exhibited spontaneous fluctuations with similar activity patterns (Supplementary Fig. 22), leading to high correlations across neuropils (Fig. 6K). After training, the model dynamics generated highly consistent functional connectivity strengths and patterns (Fig. 6J and Supplementary Fig. 23) compared to experimental data, regardless of whether during the training or testing phases.

Discussion

In conclusion, BrainTrace advances SNN online learning by jointly optimizing three axes that have rarely been achieved in concert: generality, efficiency, and usability. First, in contrast to neuromorphic training pipelines centered on simplified LIF abstractions[10–12], BrainTrace introduces two canonical modeling primitives, `AlignPre` and `AlignPost`, to support training across a broad spectrum of dynamic models that are essential for biological spiking network modeling. Second, by exploiting the inherent properties of spiking dynamics, BrainTrace formulates a theoretically guaranteed learning rule with memory scaling linear in the number of neurons while maintaining a high approximation to exact gradients. Third, inspired by modern autograd ecosystems for ANNs[21,22], BrainTrace provides an automated online-learning compiler that instantiates numerically stable, hardware-efficient updates directly from high-level model code, enabling end-to-end workflows from mathematical specification to deployment on CPUs, GPUs, and TPUs. Validation across diverse SNN architectures and tasks, including task-oriented neuromorphic benchmarks and neuroscience-oriented brain-simulation models, demonstrates that the BrainTrace system is efficient, high-performing, and scalable.

At the algorithmic level, pp-prop offers insights into a fundamental question in computational neuroscience: What advantages do SNNs offer over conventional rate-based models? While the energy efficiency of event-driven computation is well established[1], our results reveal an additional benefit in the context of learning algorithms. Approximating online gradient computations in recurrent neural networks has been extensively studied, yet existing

https://doi.org/10.1038/s41467-026-68453-w

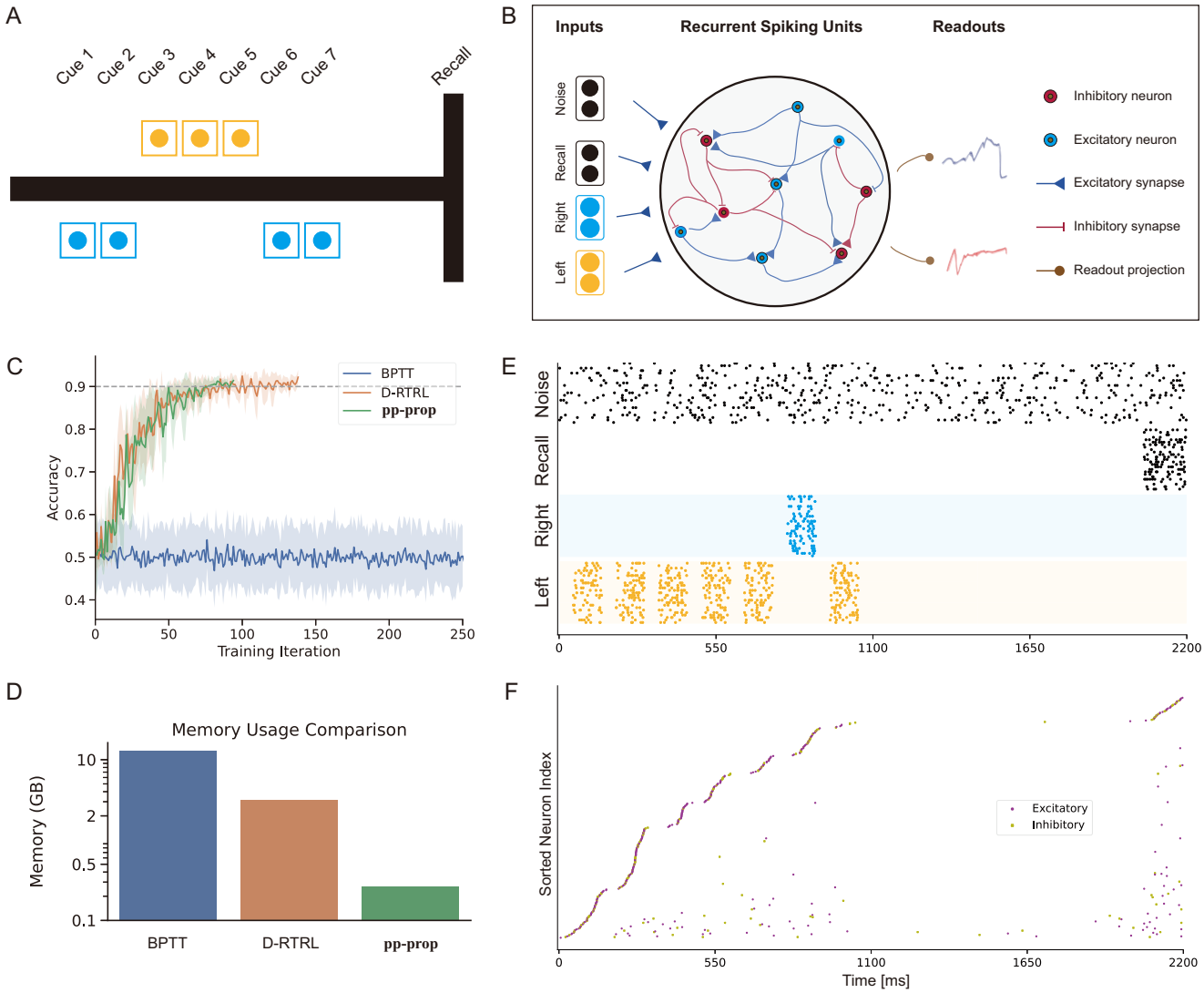


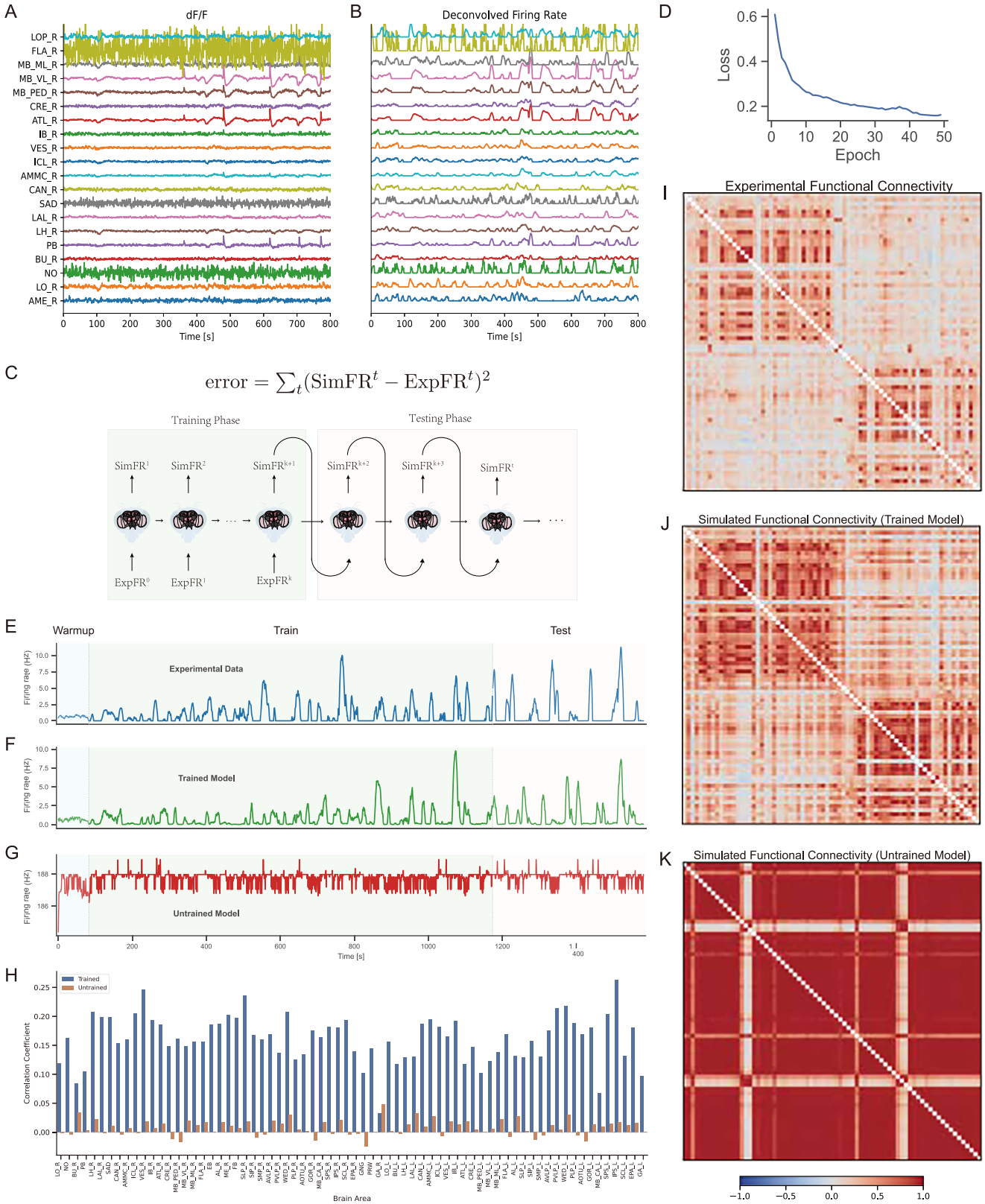
Fig. 5 | Training and analysis of biologically-informed excitatory-and-inhibitory network for cognitive task performance. A Schematic of a virtual T-maze trial. Seven visual cues are presented (three left, four right), with the majority direction (right) indicating the correct choice for reward. **B** Architecture of the recurrent excitatory-inhibitory spiking network implementing the T-maze task. The network transforms time-varying sensory inputs encoding cues and task rules into binary motor outputs representing left or right decisions. **C** Testing accuracy comparison across training algorithms: BPTT, D-RTRL, and pp-prop. **D** Device memory usage comparison during network training across algorithms. **E** Example input spike trains for the evidence accumulation task, with time on the *x*-axis and neuron index on the *y*-axis. **F** Network spiking activity following pp-prop training in response to the input pattern shown in **E**. Raster plot displays sorted recurrent network spikes after training completion.

effective approximations exhibit at least quadratic memory complexity[14] (Supplementary Note J). In contrast, pp-prop theoretically shows that the event-driven dynamics of spiking neurons naturally support accurate online gradient approximations with only linear complexity. It builds on diagonal approximation strategies[8] by exploiting the inherent sparsity of spiking activity[42,43]. Crucially, pp-prop introduces a rank-one approximation based on the consistent sign of spike-based inputs, a biologically grounded yet previously untapped property that enables reduction of memory complexity from quadratic to linear while preserving learning dynamics.

At the systems level, BrainTrace turns this insight into a practical toolchain. Existing neuromorphic online learning algorithms, restricted to simple LIF neurons, typically bypass the need for general programming frameworks. They provide predefined online learning implementations tailored to specific neuron models, which limits users to constructing networks from a fixed library of components (Supplementary Note K). In contrast, realistic brain modeling demands the flexibility to customize network dynamics and connectivity[44]. BrainTrace addresses this demand by extending the general-purpose brain modeling capabilities of BrainPy[20,44,45] with automated online learning support. This design allows researchers to focus on model development and scientific inquiry rather than low-level implementation details, an advantage analogous to the role of modern deep learning frameworks in enabling rapid prototyping[21,22].

At the application level, BrainTrace shows its unique advantage in long-horizon training of large-scale brain simulation models. As large-scale neuroscience data[46] and global connectome datasets[15,47] become increasingly available, the neuroscience community faces a pressing challenge to leverage these rich datasets to construct functional models that faithfully reproduce observed neural dynamics. Our *Drosophila* whole-brain neural activity fitting demonstrates this capability: training a 125,000+ neuron network to reproduce complex spatiotemporal activity patterns over long periods, a task previously computationally intractable in a single GPU (Supplementary Fig. 24).

https://doi.org/10.1038/s41467-026-68453-w



However, several limitations need to be solved in the future. First, while pp-prop performs robustly across benchmarks, its approximation quality can depend on firing statistics and network depth; tighter theory for accuracy-sparsity trade-offs and adaptive control of approximation rank would sharpen its operating regime. Second, extending the framework with additional plasticity mechanisms, e.g., metaplasticity and homeostatic rules[48], may improve biological fidelity and stability in long-horizon training. Third, hybrid pipelines that combine ANN-SNN conversion[49] for initialization with pp-prop for continual adaptation could merge the efficiency of converted inference with the flexibility of online learning. Fourth, task-specialized architectures and training strategies aimed at state-of-the-art performance on the most challenging neuromorphic datasets remain to be engineered. Finally, deeper

Fig. 6 | Fitting *Drosophila* resting-state neural activity by training whole-brain spiking networks. A Representative $\Delta F/F$ traces from atlas ROIs in a single fly over 800 seconds, obtained from calcium imaging experiments[17,18]. **B** Deconvolved firing rates for the corresponding neuropils shown in **A**. **C** Schematic of the training and testing paradigm. During the training phase, the model receives the experimental firing rate at time t (ExpFR^t) and predicts the rate at time t + 1 (SimFR^t) by minimizing the mean squared error. In the testing phase, the model autonomously generates sequential activations from its most recent prediction. **D** Loss convergence of whole-brain neural activity fitting using the pp-prop algorithm. Firing rate dynamics for the Mushroom body neuropil: (**E**) experimental data, (**F**) outputs from the trained model, and (**G**) outputs from an untrained control. The initial `Warmup` phase (80 seconds) initializes network activity with experimental data. In the subsequent `Train` phase, the network generates activity at time $t+1$ based on its prior output at time t. The `Test` phase evaluates the network's ability to generate activity over unseen periods. **H** Quantitative comparison of similarity between experimental data and model-generated outputs, contrasting trained versus untrained models. Functional connectivity matrices derived from (**I**) experimental recordings, (**J**) the trained whole-brain network, and (**K**) the untrained network, underscoring the efficacy of the network in capturing intrinsic connectivity patterns.

integration with emerging neuromorphic substrates[1] could unlock even greater efficiency and energy savings, particularly in low-power, embedded applications.

Methods

Inspecting inherent properties of spiking networks

Property 1: intrinsic neuronal dynamics dominate the hidden Jacobian. A typical SNN exhibits two types of recurrent connections. The first type comprises $\mathbf{D}^t$ recurrent spiking connections that propagate action potentials between neurons (see $\circledast$ in Eq. (1) and examples in Supplementary Note B). The second type $\mathbf{J}^t$ consists of intrinsic neuronal dynamics, such as leaky terms in neuronal and synaptic processes (see f Eq. (1) and examples in Supplementary Note B). 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 resulting from the element-wise intrinsic dynamics, and $\mathbf{J}^t$ is the square matrix derived from the recurrent spiking connections. $\mathbf{D}^t$ is actually a block diagonal matrix of dimensions $\mathbb{R}^{Hd\times Hd}$, consisting of H-by-H blocks. Each diagonal block, $\mathbf{D}^t_{ii}$, is a d-by-d square matrix, where d is the number of state variables aligned to neuron i. For example, refer to Supplementary Note B. Without loss of generality, all subsequent derivations assume $d = 1$, making $\mathbf{D}^t$ a diagonal matrix of dimensions $\mathbb{R}^{H\times H}$.

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[42], with only 1–2% of neurons in a cortical network active at any given moment[43,50]. 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[8] and RFLO[51], 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 (Supplementary Note B). Our findings consistently indicated that the hidden Jacobian of most SNNs 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 Fig. 2B). Notably, under biologically plausible firing regimes (fraction of spiking neurons $< 10\%$), the cosine similarity exceeds 99%.

Proposition 1. In sparsely activated SNNs, 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:

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} \approx \mathbf{D}^t. \tag{9}$$

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 2 and its proof in Supplementary Note C; this observation has also been illustrated in ref. 52 for rate-based recurrent neural networks).

Theorem 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

$$\frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t} = \mathbf{D}^t_f \otimes \mathbf{x}^t, \tag{10}$$

where $\mathbf{D}^t_f$ is a diagonal matrix with $\frac{\partial \mathbf{h}^t_i}{\partial \mathbf{I}^t_i}$ as its diagonal entry $\mathbf{D}^t_{f,ii}$, and $\otimes$ is the Kronecker product.

$\mathbf{D}^t_f$ is actually a block diagonal matrix of dimensions $\mathbb{R}^{Hd\times H}$, consisting of H-by-H blocks. Each diagonal block, $\mathbf{D}^t_{f,ii}$, is a d-by-1 matrix. For example, refer to Supplementary Note B. With loss of generality, all subsequent derivations assume $d = 1$, making $\mathbf{D}^t_f$ a diagonal matrix of dimensions $\mathbb{R}^{H\times H}$.

Property 3: sign-preserving neuronal outputs induce low-rank structure. Another unique characteristic of SNNs 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 Supplementary Eq. 63, 64); while for `AlignPre` projections, $\mathbf{x}^t$ denotes the synaptic conductance (see Supplementary Eq. 61, 62). 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 $\mathbf{D}^t_f$ 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 3 formalizes this concept, with its theoretical proof presented in Supplementary Note D and empirical validation on various spiking dynamics using real-world neuromorphic datasets provided in Supplementary Note F.

Theorem 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 $r \to \infty$:

$$\lim_{r\to\infty} \cos\left(\sum_{i=1}^r \mathbf{a}_i \otimes \mathbf{b}_i, \left(\frac{1}{r}\sum_{i=1}^r \mathbf{a}_i\right) \otimes \sum_{i=1}^r \mathbf{b}_i\right) \to 1. \tag{11}$$

Derivation of online learning algorithms

Theoretical basics of RTRL. Let us consider biological SNNs 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

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}} = \sum_{t\in\mathcal{T}} \sum_{k=1}^{t} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^k} \frac{\partial \mathbf{h}^k}{\partial \boldsymbol{\theta}^k}, \tag{12}$$

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. (12), two methods are commonly used, that is BPTT and RTRL[13], which iteratively perform the error propagation on the unfolded computation graph through time backwardly and forwardly, respectively (also refer to Supplementary Fig. 16). For the RTRL, the gradient is updated in the forward direction:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \boldsymbol{\epsilon}^t, \tag{13}$$

$$\boldsymbol{\epsilon}^t = \frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} \boldsymbol{\epsilon}^{t-1} + \frac{\partial \mathbf{h}^t}{\partial \boldsymbol{\theta}^t}, \tag{14}$$

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 $t \geq k$. Supplementary Fig. 16 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.

Derivation of the D-RTRL algorithm from RTRL. In the following, we concentrate on the scenario where the hidden state dimension $d = 1$ to develop the D-RTRL algorithm.

As stated above, the raw RTRL has a high memory complexity. By substituting Eqs. (9) and (10) into Eq. (14), we can approximate $\boldsymbol{\epsilon}^t$ as:

$$\boldsymbol{\epsilon}^t \approx \underbrace{\mathbf{D}^t}_{\approx \partial \mathbf{h}^t/\partial \mathbf{h}^{t-1}} \boldsymbol{\epsilon}^{t-1} + \underbrace{\mathbf{D}_f^t \otimes \mathbf{x}^t}_{=\partial \mathbf{h}^t/\partial \boldsymbol{\theta}^t}. \tag{15}$$

Eq. (15) 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:

$$\boldsymbol{\epsilon}^t \approx \mathbf{D}^t \boldsymbol{\epsilon}^{t-1} + \mathrm{diag}(\mathbf{D}_f^t) \otimes \mathbf{x}^t, \tag{16}$$

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. (16) can be reformulated using element-wise multiplication. Combining with Eq. (13), we can express the learning rule of the D-RTRL algorithm as:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathcal{L} &\approx \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \circ \boldsymbol{\epsilon}^t, \\ \boldsymbol{\epsilon}^t &\approx \mathrm{diag}(\mathbf{D}^t) \circ \boldsymbol{\epsilon}^{t-1} + \mathrm{diag}(\mathbf{D}_f^t) \otimes \mathbf{x}^t. \end{aligned} \tag{17}$$

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

D-RTRL exhibits a memory complexity of $\mathcal{O}(BP)$, where B represents the batch size and P denotes the number of parameters. This efficient scaling arises because the eligibility trace $\boldsymbol{\epsilon}^t$ only stores historical information for each active synaptic connection. In sparse neural networks, D-RTRL achieves additional memory savings by eliminating traces for zero-weight connections. Convolutional layers further benefit from this approach, as their parameter count is typically smaller than the number of pre- and postsynaptic neurons when using small kernels.

Derivation of the pp-prop algorithm from D-RTRL. Given the learning rule of the D-RTRL algorithm in Eq. (17), 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:

$$\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}^k} \approx \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \underbrace{\left(\prod_{i=k+1}^{t} \mathbf{D}^i \right)}_{\approx \partial \mathbf{h}^t/\partial \mathbf{h}^k} \underbrace{(\mathbf{D}_f^k \otimes \mathbf{x}^k)}_{=\partial \mathbf{h}^k/\partial \boldsymbol{\theta}^k}. \tag{18}$$

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

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}^t = \sum_{k=1}^{t} \frac{\partial \mathcal{L}^t}{\partial \boldsymbol{\theta}^k} \approx \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \sum_{k=1}^{t} \left(\prod_{i=k+1}^{t} \mathbf{D}^i \right) \left(\mathbf{D}_f^k \otimes \mathbf{x}^k \right). \tag{19}$$

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. (18) and (19).

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:

$$\nabla_{\boldsymbol{\theta}} L^t \approx \frac{\partial L^t}{\partial \mathbf{h}^t} \circ \left[\sum_{k=1}^{t} \underbrace{(\mathrm{diag}(\mathbf{D}^t) \circ \cdots \circ \mathrm{diag}(\mathbf{D}^{k+1}) \circ \mathrm{diag}(\mathbf{D}_f^k))}_{\text{postsynaptic activity}} \otimes \underbrace{\mathbf{x}^k}_{\text{presynaptic activity}} \right]. \tag{20}$$

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.

Equation. (20) 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 (see Theorem 3). This theorem allows us to approximate Eq. (20) by

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}^t \approx \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \circ \left(\frac{1}{t} \sum_{k=1}^{t} \mathbf{b}_k^t \otimes \sum_{k=1}^{t} \mathbf{x}^k \right), \tag{21}$$

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 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. (21) 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 pp-prop algorithm, whose learning rule can be expressed as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{L} &\approx \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \circ \left(\boldsymbol{\epsilon}_{\mathbf{f}}^t \otimes \boldsymbol{\epsilon}_{\mathbf{x}}^t\right),\\ \boldsymbol{\epsilon}_{\mathbf{x}}^t &= \alpha \boldsymbol{\epsilon}_{\mathbf{x}}^{t-1} + \mathbf{x}^t,\\ \boldsymbol{\epsilon}_{\mathbf{f}}^t &= \alpha\, \mathrm{diag}(\mathbf{D}^t) \circ \boldsymbol{\epsilon}_{\mathbf{f}}^{t-1} + (1-\alpha)\mathrm{diag}(\mathbf{D}_f^t),\end{aligned} \tag{22}$$

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

Generalization to hidden states with multiple variables ($d > 1$). The derivations above assume $d = 1$, but generalize straightforwardly to $d > 1$. When $d > 1$, the matrices transform into block structures as follows (see examples in Supplementary Note B):

- $\mathbf{h}^t \in \mathbb{R}^{Hd\times 1}$: A block vector with H blocks of shape $\mathbb{R}^{d\times 1}$, where block $\mathbf{h}_i$ contains the state variables of neuron i.
- $\mathbf{D}^t \in \mathbb{R}^{Hd\times Hd}$: A block diagonal matrix with H diagonal blocks of shape $d \times d$, where $\mathbf{D}_{ii}^t$ is the Jacobian between state variables of neuron i.
- $\mathbf{D}_f^t \in \mathbb{R}^{Hd\times H}$: A block diagonal matrix with H diagonal blocks of shape $d \times 1$, where $\mathbf{D}_{f,ii}^t = \partial \mathbf{h}_i^t/\partial \mathbf{I}_i^t$.
- $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t} \in \mathbb{R}^{Hd\times 1}$: A block vector with H blocks of shape $\mathbb{R}^{d\times 1}$.

The online learning framework in Eqs. (17) and (22) remain unchanged; only the $\circ$ operation differs. For $d = 1$, $\circ$ denotes element-wise scalar multiplication. For $d > 1$, $\circ$ it becomes block-wise multiplication, where each block undergoes d-dimensional matrix or inner product operations.

To illustrate, consider the learning rule for a single weight $\boldsymbol{\theta}_{ji}$ connecting neurons i and j. For D-RTRL, Eq. (17) becomes:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_{ji}}\mathcal{L} &\approx \sum_{t\in\mathcal{T}} \left\langle \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}_j^t}, \boldsymbol{\epsilon}_{ji}^t \right\rangle,\\ \boldsymbol{\epsilon}_{ji}^t &\approx \mathbf{D}_{jj}^t \boldsymbol{\epsilon}_{ji}^{t-1} + \mathbf{D}_{f,jj}^t \cdot \mathbf{x}_i{}^t,\end{aligned} \tag{23}$$

where $\mathbf{x}_i{}^t \in \mathbb{R}$ is a scalar, $\boldsymbol{\epsilon}^t \in \mathbb{R}^{Hd\times H}$ is a block matrix with $H \times H$ blocks where each $\boldsymbol{\epsilon}_{ji}^t \in \mathbb{R}^{d\times 1}$, and $\langle\cdot,\cdot\rangle : \mathbb{R}^{d\times 1}\times\mathbb{R}^{d\times 1} \to \mathbb{R}$ denotes the inner product.

Similarly, for pp-prop, Eq. (22) becomes:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_{ji}}\mathcal{L} &\approx \sum_{t\in\mathcal{T}} \left\langle \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}_j^t}, \boldsymbol{\epsilon}_{\mathbf{f},j}^t \right\rangle \cdot \boldsymbol{\epsilon}_{\mathbf{x},i}^t,\\ \boldsymbol{\epsilon}_{\mathbf{f},j}^t &= \alpha\, \mathbf{D}_{jj}^t \boldsymbol{\epsilon}_{\mathbf{f},j}^{t-1} + (1-\alpha)\mathbf{D}_{f,jj}^t,\end{aligned} \tag{24}$$

where $\boldsymbol{\epsilon}_{\mathbf{f}}^t \in \mathbb{R}^{Hd\times 1}$ is a block vector with each $\boldsymbol{\epsilon}_{\mathbf{f},j}^t \in \mathbb{R}^{d\times 1}$, and $\boldsymbol{\epsilon}_{\mathbf{x},i}^t \in \mathbb{R}$ tracks the presynaptic eligibility trace of neuron i.

In summary, extending the derivations from $d = 1$ to $d > 1$ simply replaces scalars with blocks and element-wise multiplications with block-wise multiplications. The online learning framework remains structurally identical, while gaining the ability to represent richer neuron dynamics and more complex temporal dependencies.

Learning signals in deep recurrent layers. At time t, we compute the learning signal $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}_l^t}$ at layer l using backpropagation through the stacked hidden layers. To maintain linear memory complexity, we consider two approaches based on when the top-down learning signal reaches layer l.

The first approach uses the learning signal arriving at layer l at time t, capturing instantaneous dependencies across hierarchical layers:

$$\nabla_{\boldsymbol{\theta}_l}\mathcal{L} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}_l^t} \circ \boldsymbol{\epsilon}_l^t. \tag{25}$$

The second approach uses the learning signal arriving at layer l at time $t-1$, introducing a one-step temporal delay:

$$\nabla_{\boldsymbol{\theta}_l}\mathcal{L} = \sum_{t\in\mathcal{T}} \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}_l^{t-1}} \circ \boldsymbol{\epsilon}_l^{t-1}. \tag{26}$$

Experimental details for cognitive tasks

Surrogate gradient function. The spike generation function (Supplementary Eq. 36) uses a Heaviside step function with a transition at the firing threshold υ_{th}. Since this function is non-differentiable, we employ a surrogate gradient approach[53] to enable gradient-based training.

During the forward pass, spikes are generated using the Heaviside function:

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

where $x = \mathbf{v}^t - \upsilon_{\mathrm{th}}$. During the backward pass, we approximate the derivative using a triangular surrogate[3]:

$$\mathrm{spike}'(x) = \mathrm{ReLU}(\alpha \cdot (\mathrm{width} - |x|)),$$

where $\alpha = 0.3$ controls the gradient amplitude, and width = 1.0 controls the gradient width. This produces a triangular function centered at $x = 0$, with peak value $\alpha \cdot$, width, and zero gradient for $|x| >$ width.

Weight initialization. For the networks defined in Supplementary Eqs. 70–78, we initialize input, recurrent, and readout weights using a scaled Gaussian distribution:

$$W_{ji} \sim \sqrt{\frac{s}{N_{\mathrm{in}}}}, \mathcal{N}(0,1),$$

where W_{ji} is the weight from neuron i to neuron j, N_{in} is the number of presynaptic neurons, and s is a scaling factor controlling the overall weight magnitude. This initialization helps keep activation and gradient variances well-behaved, improving training stability.

For networks with explicit excitatory/inhibitory (E/I) separation (Supplementary Eq. 81), we use the same scaled Gaussian distribution but enforce non-negative weights:

$$W_{ji} \sim \sqrt{\frac{s}{N_{\mathrm{in}}}}, |\mathcal{N}(0,1)|.$$

Taking the absolute value ensures that excitatory neurons produce only positive outgoing weights and inhibitory neurons only negative ones, in accordance with Dale's principle. This constraint preserves the desired biological structure while maintaining an appropriate weight scale.

Delayed match-to-sample task. We tested the long-term dependency learning capability of our algorithms by training a recurrent spiking network (Supplementary Eq. 73) on a delayed match-to-sample (DMTS) task[27,54]. This task comprises four phases: fixation, sample, delay, and test (Supplementary Fig. 17). During the sample and test phases, one of eight motion directions is presented for 500 ms. The delay phase (> 1000 ms) separates the sample and test, during which no stimulus is presented. The network must maintain sample direction information across this delay, then determine whether the test

stimulus matches. A leaky readout layer processes network activity during the test phase to produce the match/non-match decision.

The network receives input from 100 motion-direction-tuned neurons covering 360 degrees, simulating direction-selective cells in the visual cortex. The firing rate u_i^t of input neuron i follows a von Mises tuning curve:

$$u_i^t = A \exp\left(\kappa \cos\left(\theta - \theta_{\text{pref}}^i\right)\right), \tag{27}$$

where θ is the stimulus direction, θ_{pref}^i is the preferred direction of neuron i, $\kappa = 2.0$ is the concentration parameter, and $A = 40$ Hz is the maximum firing rate. A constant background rate $r_{\text{bg}} = 1$ Hz is maintained throughout all task phases. Spike trains are generated via a Poisson process with rate $u_i^t + r_{\text{bg}}$ during stimulus periods and r_{bg} otherwise (Supplementary Fig. 17).

Evidence accumulation task. Evidence accumulation is a fundamental paradigm for studying decision-making mechanisms. We adopted a task design from Morcos et al.[41], where a mouse navigates a virtual T-maze and encounters visual cues on both sides (Fig. 5A). At the T-junction, the mouse must choose the direction corresponding to the majority of cues, requiring it to accumulate and retain evidence from each side until the decision point.

We simulated this task using 100 input neurons divided into four groups of 25 neurons each: left-cue encoding, right-cue encoding, recall-cue encoding, and background noise. Cue and recall stimuli are modeled as Poisson spike trains at 40 Hz, with each stimulus lasting 150 ms separated by 50 ms intervals. Background neurons fire at a constant 10 Hz throughout the trial. Following the cue presentation, a 1000 ms delay precedes the recall stimulus (Fig. 5A).

The leaky rate readout. The networks in Fig. 4E–H and Fig. 5 use a leaky readout mechanism with output neurons corresponding to each task label. These neurons receive linear projections from the recurrent layer and perform decoding based on membrane potentials rather than spikes.

The readout dynamics follow:

$$\tau_{\text{out}} \frac{d\mathbf{y}}{dt} = -\mathbf{y} + W^{\text{out}} \mathbf{r}^{\text{rec}} + b^{\text{out}}, \tag{28}$$

where $\mathbf{y}$ is the output neuron activity, τ_{out} is the time constant, W^{out} and b^{out} are the weights and bias, and $\mathbf{r}^{\text{rec}}$ concatenates membrane potentials and spike activities from recurrent neurons.

Resting-state whole-brain in vivo imaging data

Our *Drosophila* whole-brain in vivo imaging data derive from experimental studies by Mann et al. (2017)[17] and Turner et al. (2021)[18]. They recorded resting-state neural activity in *Drosophila* using whole-brain calcium imaging of adult flies expressing the calcium-sensitive fluorescent protein GCaMP6s and membrane-tagged tdTomato. The central brain was exposed and imaged in vivo using resonant scanning two-photon microscopy at 3 µm isotropic resolution with 1.2 Hz temporal sampling for approximately 17 minutes per recording. They aligned functional data to a standard brain atlas using CMTK computational tools and defined regions of interest (ROIs) based on anatomical boundaries from established atlases (Ito or Branson). From each ROI, they extracted fluorescence time series, applied high-pass filtering to remove slow drift, and converted signals to $\Delta F/F$ measurements. The exemplary traces of recording $\Delta F/F$ can be obtained in Fig. 6A. We used deconvolution methods to transform these calcium imaging data into the neuropil firing rate (Supplementary Note H).

Whole-brain *Drosophila* connectome-constrained model

Network architecture. The whole-brain spiking network contains over 125,000 neurons and 50 million synaptic connections, organized into interconnected components (Supplementary Fig. 18).

Training operates on two distinct time scales. The neuropil firing rate is sampled at 1.2 Hz, with each data point representing approximately 833 ms of biological time. We denote the neuropil firing rate at time step T as Neuropil FR^T. In contrast, the spiking network simulation uses a 0.2 ms time step, yielding 4165 simulation steps per neuropil sample. We use t for simulation time and T for neuropil time.

At each simulation step t, the network processes both recurrent input I_{rec}^t (see Recurrent encoder)) and external input I_{ext}^t derived from the previous neuropil firing rate (see Input encoder). The recurrent connectivity matrix W is derived from the FlyWire connectome[15]. After completing all simulation steps within a neuropil time window, we compute each neuron's firing rate and transform it to neuropil-level firing rates for comparison with experimental data (see Neuropil firing rate readout).

Network dynamics. The network dynamics follow the LIF model developed by Shiu et al.[16]:

$$T_{\text{mbr}} \frac{dv_i}{dt} = g_i - (v_i - V_{\text{resting}}) + I_{\text{ext}}, \tag{29}$$

$$\frac{dg_i}{dt} = -\frac{g_i}{\tau} + I_{\text{rec}}, \tag{30}$$

where v_i is the membrane potential and g_i is the synaptic conductance of neuron i. When $v_i \geq V_{\text{threshold}}$, the neuron fires and resets to V_{reset}, followed by a refractory period of $T_{\text{refractory}} = 2.2$ ms.

Biophysical parameters are set to physiologically realistic values: $V_{\text{resting}} = V_{\text{reset}} = -52$ mV, $V_{\text{threshold}} = -45$ mV, $R_{\text{mbr}} = 10\,\text{K}\Omega \cdot \text{cm}^2$, $C_{\text{mbr}} = 2\,\mu\text{F} \cdot \text{cm}^{-2}$, $T_{\text{mbr}} = 20$ ms, $\tau = 5$ ms, $T_{\text{delay}} = 1.8$ ms, and $W_{\text{syn}} = 0.275$ mV.

Input encoder. The input encoder transforms neuropil firing rates from the previous time step into background input for the current time step. Specifically, the encoder takes the neuropil firing rate at time step $T-1$ (denoted as Neuropil FR^{T-1}) and generates the external input I_{ext}^T for each neuron at time step T. All firing rates are measured in Hz. During the T-th neuropil time step, neuron i receives Poisson spike noise based on the transformed firing rate $\text{FR}_{\text{ext},i}^T$. To capture temporal dependencies between different time steps, we employ a gated recurrent unit (GRU)[55] followed by a linear transformation as the encoder for the *Drosophila* whole brain model. The transformation is given by:

$$\text{FR}_{\text{ext}}^T = \text{Linear}(\text{GRU}(\text{Neuropil FR}^{T-1})) \tag{31}$$

Then, the neuropil-time-scale firing rate drives Poisson spike noise added to the membrane potential:

$$I_{\text{ext}}(t) = \sum_j \delta(t - t_j) w_i, \tag{32}$$

where $\delta(t - t_j)$ represents Poisson spike times and w_i is the learned synaptic weight for neuron i. During training, Neuropil FR^T corresponds to experimental recordings; during inference, it refers to the simulated firing rate at the last step.

Recurrent encoder. The recurrent input to neuron i at time t is:

$$I_{\text{rec},i}(t) = \sum_j \delta(t - t_j^{\text{spk}}) w_{j,i} \tag{33}$$

The synaptic weight $w_{j,i}$ is computed by scaling the FlyWire connectome strength[15] with a sign factor (+ 1 for excitatory, −1 for inhibitory) and the baseline synaptic weight W_{syn}.

Neuropil firing rate readout. Each neuropil's firing rate is computed as a weighted average of its constituent neurons' firing rates. Because neuron density and connectivity vary across neuropils, simple averaging would distort the measured activity. Instead, each neuron is weighted by its normalized synaptic projection count into that neuropil:

$$\text{SimFR}_i = \frac{\sum_j r_j w_j}{\sum_j w_j} \tag{34}$$

where r_j is the firing rate of neuron j and w_j is its normalized synaptic projection weight to neuropil i, obtained from the FlyWire connectome[15]. This weighting emphasizes neurons with stronger anatomical influence, yielding a more biologically faithful estimate of neuropil activity.

Robust automatic online learning through the Jaxpr compilation

The backpropagation algorithm owes much of its success to automatic differentiation frameworks like PyTorch[21] and TensorFlow[22], which abstract away gradient computation details. Inspired by this, we developed a general programming interface for online learning in SNNs that automatically derives eligibility traces and learning signals. Users need only define network dynamics and architecture; the interface handles all online learning computations.

Developing this framework requires addressing three challenges: (1) deriving eligibility traces, (2) generating learning signals, and (3) computing parameter gradients. We leverage Jaxpr, JAX's intermediate representation[56], to inspect computation graphs using abstract symbols. This enables us to analyze user-defined models and regenerate code that encompasses both state evolution and online learning, a process we call "online learning compilation". During online learning compilation, we can also perform comprehensive optimizations to maximize computational performance and minimize the memory footprint that we need (see Supplementary Note I).

Deriving eligibility traces. Automatic derivation of eligibility traces involves three steps.

First, we analyze hidden state groups $\mathbf{h}_l = [\mathbf{v}_l^1, \cdots, \mathbf{v}_l^d]$ within each recurrent layer to determine how state variables interact. This yields the block diagonal matrices $\mathbf{D}_l^t$ for each layer $l \in \{1, \ldots, L\}$.

Second, we analyze weight-to-hidden relationships $\frac{\partial \mathbf{h}^t}{\partial \theta^t}$ by enumerating the computation graph to determine which hidden state each parameter connects to. This yields the block diagonal matrix $\mathbf{D}_f^t$.

Third, we generate the Jaxpr representation of eligibility traces (Eqs. (4)–(8)) based on the above analysis.

Generating learning signals. We use backpropagation through deep layers to generate learning signals (Eq. (2)).

Computing Eq. (25) needs to save the gradients of intermediate variables. This is not allowed directly in JAX since hidden states computed within a function cannot be passed as function parameters. To overcome this, we employ a perturbation trick: we add a zero perturbation $\delta_{\mathbf{h}}$ to each hidden state, yielding $\mathbf{h}^t + \delta_{\mathbf{h}}$. By treating this perturbation as a function parameter, we exploit the equivalence $\frac{\partial \mathcal{L}^t}{\partial \delta_{\mathbf{h}^t}} = \frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^t}$. We then use JAX's `jax.vjp` interface to compute these derivatives and extract the result as a Jaxpr representation.

For computing Eq. (26), we directly take derivatives of hidden states at the previous time step $\frac{\partial \mathcal{L}^t}{\partial \mathbf{h}^{t-1}}$.

Computing parameter gradients. We integrate eligibility traces and learning signals using JAX's `jax.custom_vjp` to define a custom vector-Jacobian product rule. This enables weight gradients to be accessed via standard interfaces like `jax.grad` while internally applying our online learning rules.

Our implementation defines two functions: (1) a forward pass function that computes network propagation and eligibility traces, and (2) a backward pass function that computes learning signals and parameter gradients by combining eligibility traces with learning signals. See Listing S4 for an example.

The resulting interface requires only two lines of code:

```
# define online computation using the pp-prop algorithm
model = braintrace.pp_prop(net, decay_or_rank=0.98)
# define online computation using the D-RTRL algorithm
model = braintrace.D_RTRL(net)
# compile the online learning graph using the input data
model.compile_graph(inputs)
```

Data availability

The datasets used in this study are publicly available and open source. The N-MNIST dataset[26] is freely available at https://www.garrickorchard.com/datasets/n-mnist. The SHD dataset[25] is publicly available at https://zenkelab.org/resources/spiking-heidelberg-datasets-shd. The IBM DVS Gesture dataset[24] 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 publicly available GitHub repository https://github.com/chaobrain/braintrace-snn-experiments[57]. The whole-brain calcium imaging data of *Drosophila*[18] can be obtained at https://doi.org/10.6084/m9.figshare.13349282.

Code availability

BrainTrace is distributed via the PyPI package index (https://pypi.org/project/braintrace) and publicly released on GitHub (https://github.com/chaobrain/braintrace) under the license of Apache License v2.0. Its documentation is hosted on the free documentation hosting platform Read the Docs (https://braintrace.readthedocs.io/). BrainTrace can be used in Windows, macOS, and Linux operating systems. The code to reproduce the experimental evaluations of Figs. 4, 5, and Table 1 is publicly available from the GitHub repository https://github.com/chaobrain/braintrace-snn-experiments[57]. The code for the modeling of Fig. 6 is publicly available from the GitHub repository https://github.com/chaobrain/fitting_drosophila_whole_brain_spiking_model[58]. The code for the SHD dataset evaluation is available at https://github.com/chaobrain/braintrace-shd-experiments[59].

References

1. Roy, K., Jaiswal, A. & Panda, P. Towards spike-based machine intelligence with neuromorphic computing. *Nature* **575**, 607–617 (2019).
2. Yamazaki, K., Vo-Ho, V.-K., Bulsara, D. & Le, N. Spiking neural networks and their applications: A review. *Brain Sci.* **12**, 863 (2022).
3. Neftci, E. O., Mostafa, H. & Zenke, F. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* **36**, 51–63 (2019).
4. Bellec, G., Salaj, D., Subramoney, A., Legenstein, R. & Maass, W. Long short-term memory and learning-to-learn in networks of spiking neurons. *Adv. Neural Inf. Process. Syst.* **31**, 787–797 (2018).
5. Huh, D. & Sejnowski, T. J. Gradient descent for spiking neural networks. *Adv. Neural Inf. Process. Syst.* **31**, 1433–1443 (2018).
6. Mehonic, A. & Kenyon, A. J. Brain-inspired computing needs a master plan. *Nature* **604**, 255–260 (2022).
7. Lobo, J. L., Del Ser, J., Bifet, A. & Kasabov, N. Spiking neural networks and online learning: An overview and perspectives. *Neural Netw.* **121**, 88–100 (2020).

8. Bellec, G. et al. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun.* **11**, 3625 (2020).
9. Bohnstingl, T., Woźniak, S., Pantazi, A. & Eleftheriou, E. Online spatio-temporal learning in deep neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **34**, 8894–8908 (2022).
10. Xiao, M., Meng, Q., Zhang, Z., He, D. & Lin, Z. Online training through time for spiking neural networks. *Adv. Neural Inf. Process. Syst.* **35**, 20717–20730 (2022).
11. Summe, T. M., Schaefer, C. J. & Joshi, S. Estimating post-synaptic effects for online training of feed-forward SNNS. *2024 International Conference on Neuromorphic Systems*, 264–271 (2024).
12. Jiang, H., De Masi, G., Xiong, H. & Gu, B. Ndot: neuronal dynamics-based online training for spiking neural networks. *Forty-first International Conference on Machine Learning* (2024).
13. Williams, R. J. & Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.* **1**, 270–280 (1989).
14. Marschall, O., Cho, K. & Savin, C. A unified framework of online learning algorithms for training recurrent neural networks. *J. Mach. Learn. Res.* **21**, 1–34 (2020).
15. Dorkenwald, S. et al. Neuronal wiring diagram of an adult brain. *Nature* **634**, 124–138 (2024).
16. Shiu, P. K. et al. A Drosophila computational brain model reveals sensorimotor processing. *Nature* **634**, 210–219 (2024).
17. Mann, K., Gallen, C. L. & Clandinin, T. R. Whole-brain calcium imaging reveals an intrinsic functional network in Drosophila. *Curr. Biol.* **27**, 2389–2396 (2017).
18. Turner, M. H., Mann, K. & Clandinin, T. R. The connectome predicts resting-state functional connectivity across the Drosophila brain. *Curr. Biol.* **31**, 2386–2394 (2021).
19. Brette, R. et al. Simulation of networks of spiking neurons: a review of tools and strategies. *J. Comput. Neurosci.* **23**, 349–398 (2007).
20. Wang, C. et al. A differentiable brain simulator bridging brain simulation and brain-inspired computing. *International Conference on Learning Representations* (2024).
21. Paszke, A. et al. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **32**, 8026–8037 (2019).
22. Abadi, M. et al. TensorFlow: a system for large-scale machine learning. *12th USENIX symposium on operating systems design and implementation*, 265–283 (2016).
23. Frostig, R., Johnson, M. J. & Leary, C. Compiling machine learning programs via high-level tracing. *Syst. Machine Learn.* **4** (2018).
24. Amir, A. et al. A low-power, fully event-based gesture recognition system. *Proceedings of the IEEE conference on computer vision and pattern recognition,* 7243–7252 (2017).
25. Cramer, B., Stradmann, Y., Schemmel, J. & Zenke, F. The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **33**, 2744–2757 (2020).
26. Orchard, G., Jayawant, A., Cohen, G. K. & Thakor, N. Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* **9**, 437 (2015).
27. Chudasama, Y. Delayed (non) match-to-sample task. *Encyclopedia of Psychopharmacology*, 372–372 (2010).
28. Motta, A. et al. Dense connectomic reconstruction in layer 4 of the somatosensory cortex. *Science* **366**, eaay3134 (2019).
29. Bittar, A. & Garner, P. N. A surrogate gradient spiking baseline for speech command recognition. *Front. Neurosci.* **16**, 865897 (2022).
30. Subramoney, A., Nazeer, K. K., Schöne, M., Mayr, C. & Kappel, D. Efficient recurrent architectures through activity sparsity and sparse back-propagation through time. *International Conference on Learning Representations* (2022).
31. Quintana, F. M. et al. Etlp: Event-based three-factor local plasticity for online learning with neuromorphic hardware. *Neuromorph. Comput. Eng.* **4**, 034006 (2024).
32. Ortner, T., Pes, L., Gentinetta, J., Frenkel, C. & Pantazi, A. Online spatio-temporal learning with target projection. *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems*, 1–5 (2023).
33. Apolinario, M. P. E. & Roy, K. S-TLLR: STDP-inspired temporal local learning rule for spiking neural networks. *Trans. Mach. Learn. Res.* **10**, 1–40 (2025).
34. Yin, B., Corradi, F. & Bohté, S. M. Accurate online training of dynamical spiking neural networks through forward propagation through time. *Nat. Mach. Intell.* **5**, 518–527 (2023).
35. Hammouamri, I., Khalfaoui-Hassani, I. & Masquelier, T. Learning delays in spiking neural networks using dilated convolutions with learnable spacings. *International Conference on Learning Representations* (2024).
36. Zhu, Y., Ding, J., Huang, T., Xie, X. & Yu, Z. Online stabilization of spiking neural networks. *International Conference on Learning Representations* (2024).
37. Fang, W. et al. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. *Proceedings of the IEEE/CVF international conference on computer vision,* 2661–2671 (2021).
38. Mihalaş, Ş. & Niebur, E. A generalized linear integrate-and-fire neural model produces diverse spiking behaviors. *Neural Comput.* **21**, 704–718 (2009).
39. Vogels, T. P. & Abbott, L. F. Signal propagation and logic gating in networks of integrate-and-fire neurons. *J. Neurosci.* **25**, 10786–10795 (2005).
40. Van Vreeswijk, C. & Sompolinsky, H. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* **274**, 1724–1726 (1996).
41. Morcos, A. S. & Harvey, C. D. History-dependent variability in population dynamics during evidence accumulation in cortex. *Nat. Neurosci.* **19**, 1672–1681 (2016).
42. Attwell, D. & Laughlin, S. B. An energy budget for signaling in the grey matter of the brain. *J. Cereb. Blood Flow. Metab.* **21**, 1133–1145 (2001).
43. Kloppenburg, P. & Nawrot, M. P. Neural coding: sparse but on time. *Curr. Biol.* **24**, R957–R959 (2014).
44. Wang, C. et al. Brainpy, a flexible, integrative, efficient, and extensible framework for general-purpose brain dynamics programming. *elife* **12**, e86365 (2023).
45. Wang, C., He, S., Luo, S., Huan, Y. & Wu, S. Integrating physical units into high-performance AI-driven scientific computing. *Nat. Commun.* **16**, 3609 (2025).
46. Koch, C. & Jones, A. Big science, team science, and open science for neuroscience. *Neuron* **92**, 612–616 (2016).
47. MICrONS Consortium Functional connectomics spanning multiple areas of mouse visual cortex. *Nature* **640**, 435–447 (2025).
48. Morrison, A., Diesmann, M. & Gerstner, W. Phenomenological models of synaptic plasticity based on spike timing. *Biol. Cybern.* **98**, 459–478 (2008).
49. Bu, T. et al. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. *International Conference on Learning Representations* (2022).
50. Olshausen, B. A. & Field, D. J. Sparse coding of sensory inputs. *Curr. Opin. Neurobiol.* **14**, 481–487 (2004).
51. Murray, J. M. Local online learning in recurrent networks with random feedback. *Elife* **8**, e43299 (2019).
52. Mujika, A., Meier, F. & Steger, A. Approximating real-time recurrent learning with random Kronecker factors. *Adv. Neural Inf. Process. Syst.* **31**, 6594–6603 (2018).
53. Zenke, F. & Ganguli, S. Superspike: Supervised learning in multilayer spiking neural networks. *Neural Comput.* **30**, 1514–1541 (2018).
54. Daniel, T. A., Katz, J. S. & Robinson, J. L. Delayed match-to-sample in working memory: A brainmap meta-analysis. *Biol. Psychol.* **120**, 10–20 (2016).

55. Cho, K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Conference on Empirical Methods in Natural Language Processing* (2014).
56. Bradbury, J. et al. JAX: composable transformations of Python +NumPy programs http://github.com/google/jax (2018).
57. Wang, C. chaobrain/braintrace-snn-experiments: Release version 0.1 https://doi.org/10.5281/zenodo.17847403 (2025).
58. Wang, C. chaobrain/fitting_drosophila_whole_brain_spiking_model: Release version 0.1 https://doi.org/10.5281/zenodo.17892849 (2025).
59. Wang, C. & Msra Xmq. chaobrain/braintrace-shd-experiments: Release version 0.1 https://doi.org/10.5281/zenodo.17791424 (2025).
60. Yin, Y., Chen, X., Ma, C., Wu, J. & Tan, K. C. Efficient online learning for networks of two-compartment spiking neurons. *2024 International Joint Conference on Neural Networks*, 1–8 (2024).
61. Nowotny, T., Turner, J. P. & Knight, J. C. Loss shaping enhances exact gradient learning with eventprop in spiking neural networks. *Neuromorph. Comput. Eng.* **5**, 014001 (2025).
62. Samadzadeh, A., Far, F. S. T., Javadi, A., Nickabadi, A. & Chehreghani, M. H. Convolutional spiking neural networks for spatio-temporal feature extraction. *Neural Process. Lett.* **55**, 6979–6995 (2023).
63. Sun, H. et al. A synapse-threshold synergistic learning approach for spiking neural networks. *IEEE Trans. Cogn. Dev. Syst.* **16**, 544–558 (2023).

Acknowledgements

This work was supported by the Young Scientists Fund of the National Natural Science Foundation of China (No. 3240070449, C.M.W.) and the National Natural Science Foundation of China (No. T2421004, S.W.).

Author contributions

Conceptualization and Methodology: C.M.W., X.S.D., J.D.J., S.W. Software and Investigation: C.M.W. Analysis: C.M.W., X.S.D., Z.L.J., M.Q.X., J.D.J., X.L. Theorem Proof: X.S.D., C.M.W. Visualization: C.M.W., X.L., Z.L.J. Writing: C.M.W., Z.L.J., S.W. Writing (Review & Editing): C.M.W., Z.L.J., M.Q.X., X.S.D., J.D.J., X.L., SW. Resources: C.M.W., Y.X.H., S.W. Funding Acquisition: C.M.W., S.W.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41467-026-68453-w.

Correspondence and requests for materials should be addressed to Chaoming Wang or Si Wu.

Peer review information *Nature Communications* thanks Thomas Nowotny and the other anonymous reviewer(s) for their contribution to the peer review of this work. A peer review file is available.