Supplementary Information for Adaptive spatiotemporal neural networks through complementary hybridization

Yujie Wu[1, 2, 5, 6], **Bizhao Shi**[3, 4, 6], **Zhong Zheng**[1], **Hanle Zheng**[1], **Fangwen Yu**[1], **Xue Liu**[1], **Guojie Luo**[3, 4], **Lei Deng**[1✉]

Contents

List of Figures

List of Tables

[1]Center for Brain Inspired Computing Research (CBICR), Department of Precision Instrument, Tsinghua University, Beijing, China.
[2]Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR.
[3]School of Computer Science, Peking University, Beijing, China.
[4]Center for Energy-Efficient Computing and Applications, Peking University, Beijing, China.
[5]Institute of Theoretical Computer Science, Graz University of Technology, Graz, Austria.
[6]These authors contributed equally to this work.
✉Email: leideng@mail.tsinghua.edu.cn

Supplementary Section 1 Experimental parameter setting

In this section, we elaborate on the model architectures and parameters, and we provide specific configurations as listed in Supplementary Table 1. For directly-hybrid models, neuron types within each layer are allocated based on a predetermined SNN ratio. For instance, with an SNN ratio of 0.4, the network structure for directly-hybrid models with 400 neurons per hidden layer applied to the S-MNIST dataset would follow the configuration [input-(240,160)-(240,160)-10].

For the deep-learning-oriented datasets, including PTB and S-MNIST, we establish HSTNNs with two hybrid hidden layers based on vanilla RNNs and LIF SNNs and use Cross Entropy (CE) as the loss function. For PTB, the word embedding size is 650, and the output dimension is 10000. During the training process, we halve the learning rate every 25 epochs. The SGD optimizer is applied for PTB with the momentum equal to 0.9. For S-MNIST, images are fed into networks in a column-wise manner. Note that the output spikes of the last hidden layer are accumulated across time steps.

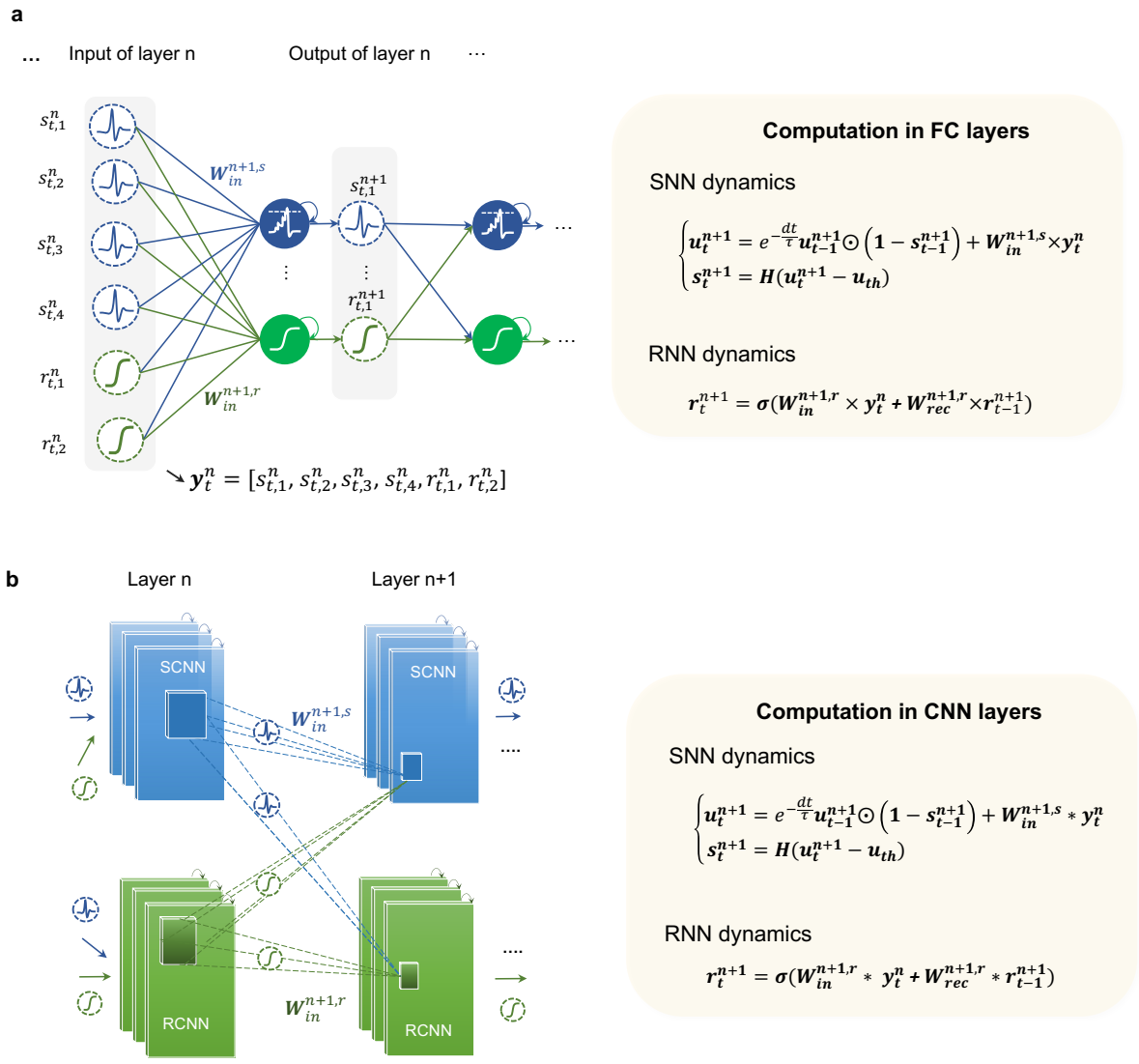
Neuromorphic datasets comprise event-driven spike trains. To accelerate the training, we follow the setup in prior work[1–3] and preprocess the spike inputs by compressing the events in a time interval (i.e., temporal resolution dt) and reducing the input size (i.e., spatial resolution, ds). In our experiments, $dt = 3$ms and $ds = 1$ are adopted on N-MNIST, $dt = 20$ms, and $ds = 4$ on DVS-Gesture. A convolutional neural network with the structure of [input-128C3-AP2-256C3-AP2-384C3-AP2-FC256-output] is built for the two datasets. The 128C3 denotes the convolutional layer with the kernel size of 3 and the number of output channels of 128, AP2 denotes the average pooling with the pooling size of 2, and FC denotes the fully connected layer. Besides, an FC-based network architecture with two hybrid hidden layers is also deployed on these two datasets. The loss functions are all set as Mean Square Error (MSE). For CIFAR10-DVS, we follow the high-performance architecture from reference[4]. It integrates the event data into 10 frames per sample and downscales each event image to $48\times48\times2$ before being fed into the network. In addition, a 13-layer convolutional neural network with the structure of [input-128C3-128C3-AP2-128C3-128C3-AP2-256C3-256C3-AP2-512C3-512C3-512C3-512C3-10] is applied for CIFAR10-DVS. The SGD optimizer, with a momentum of 0.9, and a loss function comprising a weighted average of CE and MSE functions, with factors for the CE and MSE components set at 0.95 and 0.05, respectively, are employed. For the LIF-based SCNN models, we adhere to the modelling approach of reference[4], where a triangular surrogate function is utilized.

Supplementary Table 1. Parameter configurations for sequence learning tasks.

Parameter	Deep-learning-oriented		Neuromorphic-computing-oriented		
Dataset	PTB	S-MNIST	N-MNIST	DVS-Gesture	CIFAR10-DVS
Network architecture	FC	FC	CNN | FC	CNN | FC	CNN
Number of neurons per hidden layer	650	400 / 800	- | 800	- | 400	-
Value of k in the surrogate function	1.0	1.0	0.5 | 0.5	1.0 | 0.5	-
Decay of membrane potential	0.6	0.6	0.3 | 0.3	0.3 | 0.3	0.4
Spiking threshold	0.6	0.6	0.3 | 0.3	0.5 | 0.3	1.0
Number of simulation steps	35	28	15 | 15	60 | 60	10
Output dimensions	10000	10	10 | 10	11 | 11	10
Batch size	25	100	100 | 100	36 | 72	72
Learning rate (Adaptation stage)	0.5	0.0003	0.0008 | 0.0003	0.0005 | 0.0008	0.05
Learning rate (Restoration stage)	0.1	0.0003	0.0008 | 0.0003	0.0003 | 0.0008	0.05
Number of epochs (Adaptation stage)	200	150	100 | 150	320 | 320	300
Number of epochs (Restoration stage)	200	150	150 | 200	320 | 320	300

Supplementary Section 2 Illustration of information flow in the hybrid model

We show in Supplementary Figure 1 the hybrid information flow of input and output dimensions in both fully connected (FC) and convolutional (Conv) architectures. Particularly, in panel (a), we show four spiking inputs, two non-spiking inputs, and two neurons with one in each group for simple illustration. The main difference between the two architectures lies in the computational operations, where FC layers employ matrix multiplications (denoted by $\times$) and Conv layers employ convolutional operations (denoted by $*$). Since the neural dynamics of RNN and SNN groups are independent, the internal network structures and models can be configured flexibly without any restrictions.



Supplementary Figure 1. A comprehensive illustration of the hybrid information flow in a fully connected (FC) architecture (a) and the convolutional (Conv) architecture (b). For demonstration, we consider four spiking inputs and two non-spiking inputs from the previous layer in panel (a). The main difference between FC and Conv architectures lies in the computational operations, where FC layers employ weight matrix multiplications denoted by $\times$ and Conv layers employ convolutional operations denoted by $*$. For the input manner, the use of hybrid inputs in each layer is the same for different network architectures.

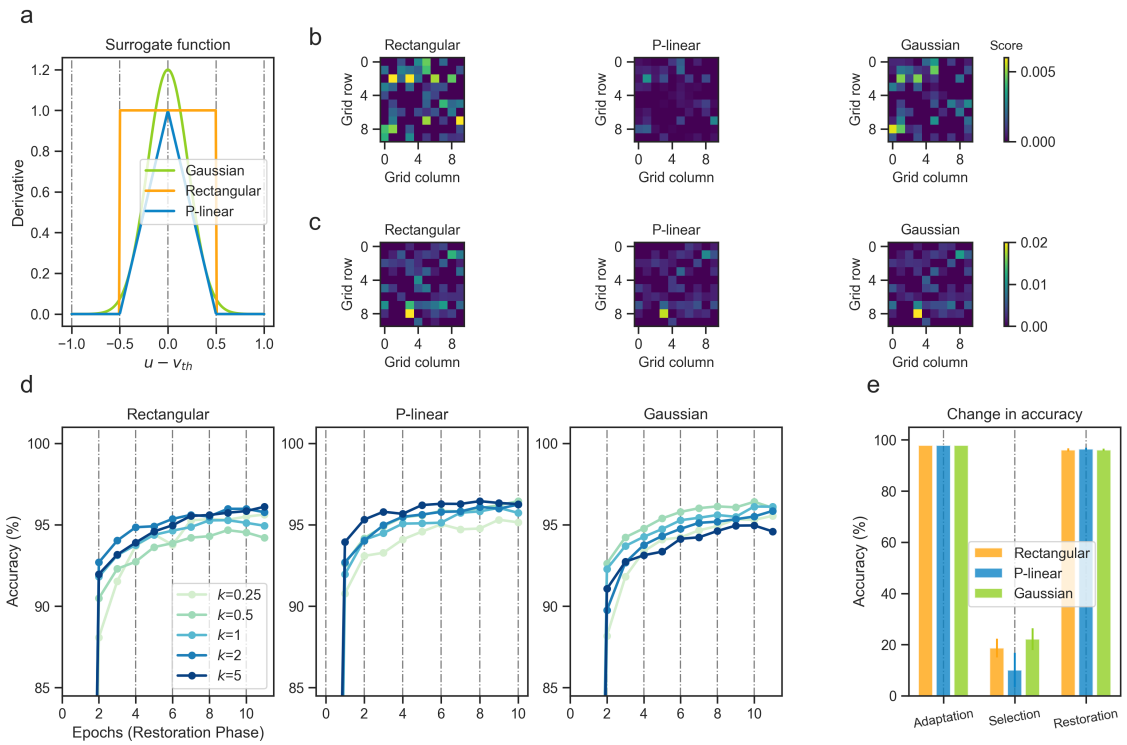
Supplementary Section 3 Influence of surrogate functions

As depicted in Supplementary Figure 2(a), we analyze the impact of three typical surrogate function formats on the task performance, including the rectangular function F_1, the Gaussian function F_2, and the piecewise linear function F_3, which can be formulated as

$$F_1(u-u_{th}) := \begin{cases} \frac{1}{k}, & |u-u_{th}| \leq \frac{k}{2} \\ 0, & otherwise. \end{cases}, \quad F_2(u-u_{th}) := \frac{1}{\sqrt{2\pi k}} e^{-\frac{(u-u_{th})^2}{2k}}, \quad F_3(u-u_{th}) := \begin{cases} \frac{2}{k}(1-\frac{|u-u_{th}|}{k}), & |u-u_{th}| \leq k \\ 0, & otherwise. \end{cases}. \tag{1}$$

We set $k = 1$ in Supplementary Figure 2(b)-(c) and (e).

We applied these functions during the learning of HSTNNs while maintaining the same initial hyper-parameters across three learning stages. The S-MNIST was used for comparison. We built the HSTNNs with the structure of [28-100-100-10] in this experiment. The results shown in Supplementary Figure 2 suggest that the format of the surrogate function can affect the importance scores of neurons, which in turn could influence the selection patterns of important neurons in the *Selection* stage. In the meantime, we found an obvious similarity among the distributions of importance scores under different surrogate functions, which implies their insignificant accuracy differences after the *Restoration* stage. Furthermore, we found different surrogate functions would influence the selection of the optimal k hyper-parameter that searches for a balance between the gradient passing width and approximation error during backpropagation.



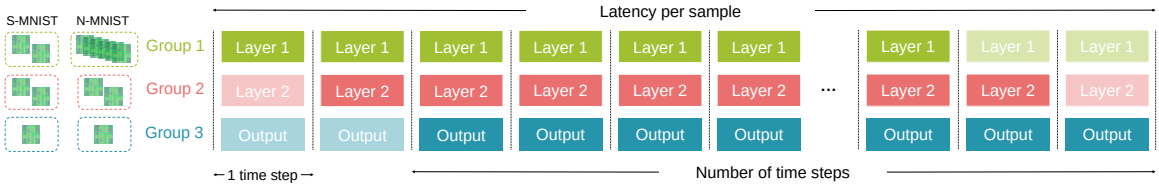
Supplementary Figure 2. Influence of the surrogate function on the performance of HSTNN over the S-MNIST dataset. (a) Different formats of surrogate functions, including the rectangular function, the piece-wise linear (P-linear) function, and the Gaussian function defined in Equations (1) for comparison. (b)-(c), Importance scores of neurons in the first hidden layer of HSTNN with (b) 0.95 and (c) 0.05 SNN ratios under different surrogate functions. (d) Influence of the hyper-parameter k of surrogate functions on the model performance. Curves represent testing accuracy in the *Restoration* stage. (e) Comparison of testing accuracy of HSTNNs with different surrogate functions in three training stages. The error bars represent the standard derivation over five runs.

Supplementary Section 4 Details of implementation on neuromorphic hardware

a

	S-MNIST					N-MNIST				
SNN ratio	0	0.25	0.5	0.75	1	0	0.25	0.5	0.75	1
Number of cores	5					11				
Number of time steps	28					15				
Latency per time step (μs)	3.84	2.61	2.44	1.97	2.20	7.10	5.83	5.70	4.88	4.90
Energy per sample (μJ)	10.61	8.30	6.89	5.41	4.83	30.00	28.51	27.71	27.03	26.16

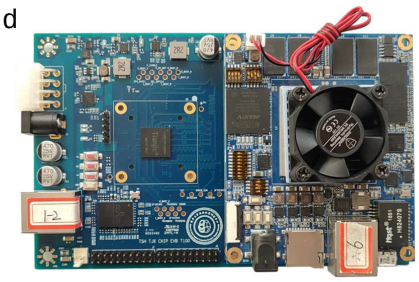
b



c

Operation	Input data type	Output data type
$\mathbf{V} = \mathbf{WX} + \mathbf{B}$	W: INT8 X: INT8	INT32
$\mathbf{V} = \mathbf{\Sigma X} + \mathbf{B}$	INT8	INT32
Memory data move	INT8/INT32	INT8/INT32
LIF dynamics	INT32	Spike: INT8
Lookup table	INT8	INT8
Data transmission	INT8/INT32	INT8/INT32

d

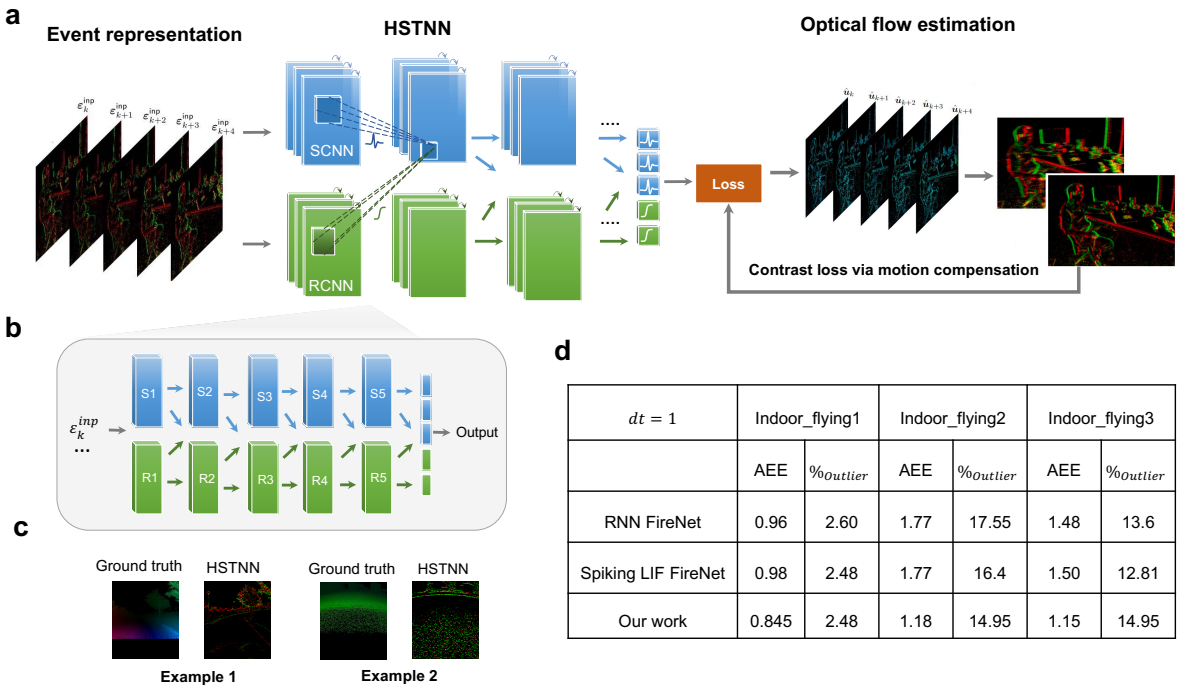


Supplementary Figure 3. Details of implementation on neuromorphic hardware. (a) Detailed experimental results of HSTNNs with different SNN ratios executed on the chip. (b) Mapping layers of the networks to different core groups with pipelined parallel computation. (c) Main chip operations used in deploying HSTNNs. (d) The TianjicX development board is used for hardware experiments.

Supplementary Section 5 Applying the hybrid model to optical flow estimation

We apply our hybrid model to the optical flow estimation task on the Multi Vehicle Stereo Event Camera (MVSEC) dataset, following the experimental setups and parameter configurations of prior work[5].

Our event stream is segmented into partitions, each containing an equal number of events, to form event-based images that are sequentially fed into the network. The network predicts an optical flow map for every pixel and polarity in each partition, correlating each input event with a motion vector. We utilize the contrast maximization loss[6] for the backward pass after processing a sufficient number of events.



$dt = 1$	Indoor_flying1		Indoor_flying2		Indoor_flying3	
	AEE	$\%_{Outlier}$	AEE	$\%_{Outlier}$	AEE	$\%_{Outlier}$
RNN FireNet	0.96	2.60	1.77	17.55	1.48	13.6
Spiking LIF FireNet	0.98	2.48	1.77	16.4	1.50	12.81
Our work	0.845	2.48	1.18	14.95	1.15	14.95

Supplementary Figure 4. Illustration of the optical flow estimation task. (a) Optical flow pipeline for constructing the hybrid model, adapted from Figure 1 in reference[5]. (b) The network structure was adapted from the original FireNet[5,7], which is composed of three spiking and non-spiking recurrent convolutional layers (R1&S1, R2&S2 and R4&S4), two spiking and non-spiking recurrent ResNet blocks (R3&S3 and R5&S5), and a final prediction layer. Each spiking or non-spiking layer has a single stride, 32 output channels, and employs 3×3 kernel encoders. Neurons in these layers follow the corresponding neural dynamics described by Equations (6), (7), and (15) in reference[5]. Each spiking (or non-spiking) residual block contains two spiking (or non-spiking) convolutional layers with skip connections, aiming to preserve information and prevent gradient vanishing in the motion estimation[7]. The final prediction layer has a single stride and two output channels, which integrates spiking representations with non-spiking components to estimate motion. (c) Qualitative evaluation of warped input events by the motion compensation using outputs of the HSTNN. (d) Quantitative results including average endpoint error (AEE), and the percentage of points with AEE greater than 3 pixels and 5% of the optical flow vector magnitude (%Outliers), alongside results from single-paradigm models in reference[5]. We use the neuron models as described in Equations (6), (7), and (15) in reference[5], and construct the HSTNN with an SNN ratio of 0.5 for comparison.

Our hybrid model preserves the core architecture from recent FireNet[5,7], which is composed of three spiking (or non-spiking) convolutional layers, two recurrent (or spiking) ResNet blocks, and a final prediction layer, as illustrated in Supplementary Figure 4(a).

Each spiking or non-spiking convolutional layer has a single stride, 32 output channels, and employs 3×3 kernel encoders. Neurons in these layers follow the corresponding spiking or non-spiking dynamics described by Equations (6), (7), and (15) in reference[5]. The residual block contains two recurrent spiking (or non-spiking)

convolutional layers with skip connections, aiming to preserve information and prevent gradient vanishing in the motion estimation[7]. The final prediction layer has a single stride, two output channels, which integrates spiking representations with non-spiking components to estimate motion. In line with the setup in[5], HSTNNs compress a fixed number of events ($N_e = 1,000$) into a tensor-represented event image, including per-pixel and per-polarity information. This tensor is fed into the RCNN and SCNN modules, with outputs from each module concatenated and passed to subsequent layers, also shown in Supplementary Figure 4(b). Network parameters are optimized by minimizing the contrast maximization proxy loss[2], aiming to align events accurately to minimize motion blur after deblurring with the predicted motion. The overall training function is a compound loss, combining contrastive loss with a local smoothness regularizer[8], described as:

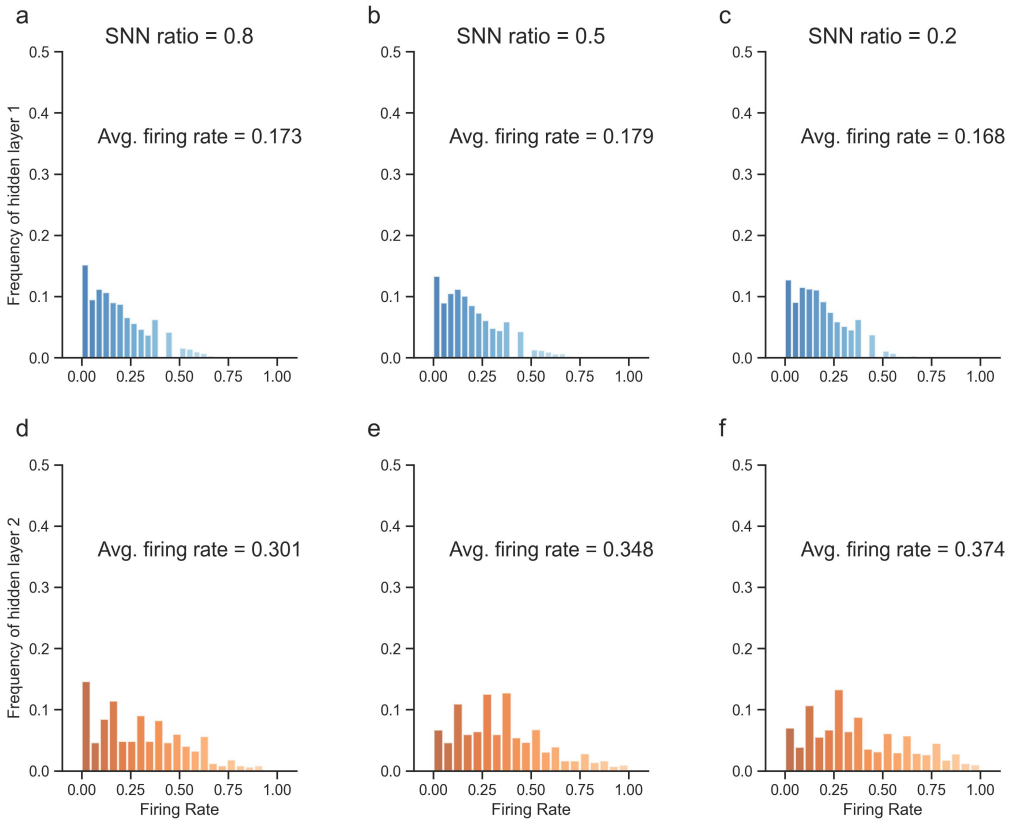
$$L_{\text{overall}} = L_{\text{contrastive}} + \lambda L_{\text{smooth}}, \tag{2}$$

where λ balances the two losses' effects ($\lambda = 0.001$ in our experiments). Further modelling details can be found in[5].

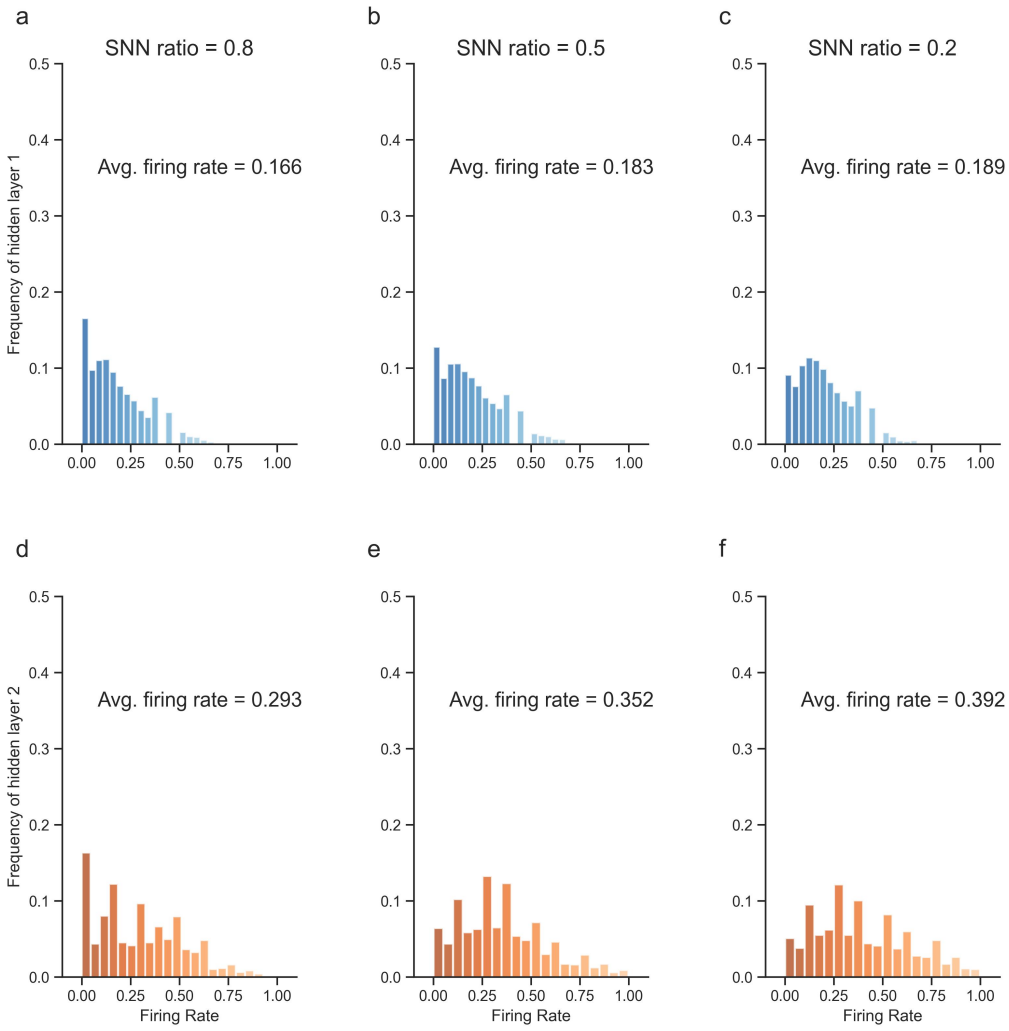
Supplementary Figure 4(c)-(d) provides the quantitative and qualitative evaluations for the optical flow estimation by HSTNNs. We employed here average endpoint error (AEE) and the percentage of points with AEE greater than 3 pixels and 5% of the magnitude of the optical flow vector (%Outlier) as performance metrics, following references[2,5]. Lower AEE and %Outlier values indicate better performance. As can be seen from Supplementary Figure 4(c), HSTNNs generally outperform the single-paradigm RNNs and spiking LIF networks reported by the work[5], demonstrating the applicability and effectiveness of the proposed hybrid methodology.

Supplementary Section 6 Visualization of spiking activities in HSTNNs

To examine the neural states within the HSTNN, we quantified the spiking activities of the network after the three-stage learning process. We tracked the average population firing rate within each hidden layer across a set of randomly selected testing samples from both S-MNIST and N-MNIST datasets. Supplementary Figures 5 and 6 suggest that the spiking activities within the HSTNN can be influenced by the ratio of spiking neurons. Specifically, as the SNN ratio decreases, there is a general increase in the population firing rate on both datasets, with a more pronounced effect observed in the second hidden layer. This is possibly due to the reduction of the overall number of spiking neurons. Since different dynamics and coding schemes of spiking neurons and continuous neurons encode information in different ways, as the decrease of SNN ratio, the HSTNN may tend to activate sufficient spiking neurons to abstract features from the input data flow, resulting in an increase in the firing rate in a reduced population size of spiking neurons.



Supplementary Figure 5. Histogram of spiking activities of the learnt HSTNN across 200 randomly selected testing samples from the S-MNIST dataset. (a)-(c) The frequency of spiking activities in the first hidden layer, evaluated at three different SNN ratios. (d)-(f) The frequency of spiking activities in the second hidden layer, assessed at three different SNN ratios.



Supplementary Figure 6. Histogram of spiking activities of the learnt HSTNN across 200 randomly selected testing samples from the N-MNIST dataset. (a)-(c) The frequency of spiking activities in the first hidden layer, evaluated at three different SNN ratios. (d)-(f) The frequency of spiking activities in the second hidden layer, assessed at three different SNN ratios.

Supplementary Section 7 Automatic search for the optimal SNN ratio

In an HSTNN, the SNN ratio plays a crucial role in determining the balance between the functional performance and the computational cost, making it a key factor in achieving efficient and accurate spatiotemporal data processing. An open issue is how to optimize the SNN ratio for adapting to different application scenarios. Here we present a method to automatically optimize the SNN ratio in HSTNNs, enabling users to configure the model according to their specific requirements.

In order to meet the diverse requirements across various real-world applications and enable flexible optimization, we formalize this problem as an optimization problem by using a weighted loss function that combines the impacts of task performance and the computational cost. The overall loss function can be expressed as:

$$\mathrm{L}_{\mathrm{overall}}(r) = \lambda_1 \mathrm{L}_{\mathrm{cost}}(r) + \lambda_2 \mathrm{L}_{\mathrm{P}}(r), \tag{3}$$

where λ_1 and λ_2 are the weighted coefficients. $\mathrm{L}_{\mathrm{cost}}(r)$ represents the normalized computational cost for the HSTNN with an SNN ratio r, which can be calculated by the cost estimate for HSTNN in Equation (25) . $\mathrm{L}_{\mathrm{P}}(r)$ represents the corresponding normalized functional performance, which can be instantiated as accuracy or negative perplexity. This loss function can be defined as the normalized difference between the functional performance of the HSTNN ($P_{\mathrm{HSTNN}}(r)$) and the minimum performance between the RNN (P_{RNN}) and SNN models (P_{RNN}). The mathematical expression is

$$\mathrm{L}_{\mathrm{P}}(r) = \frac{\min(P_{\mathrm{RNN}}, P_{\mathrm{SNN}}) - P_{\mathrm{HSTNN}}(r)}{|P_{\mathrm{RNN}} - P_{\mathrm{SNN}}|}. \tag{4}$$

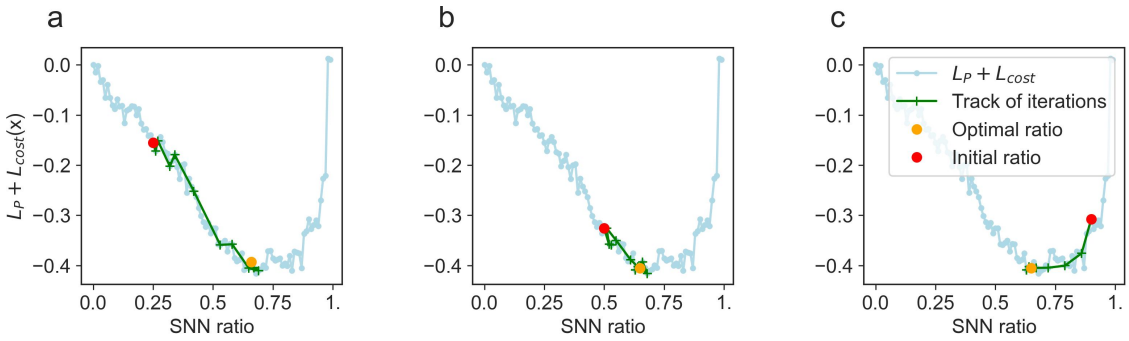
By developing the weighted loss function $\mathrm{L}_{\mathrm{overall}}(r)$, users only need to customize the weighted coefficients according to their specific requirements. We expect that this design allows them to obtain different optimal SNN ratiosthat are more suitable for different application scenarios.

Given the expression of the optimization problem, the remaining issue is to find the values of the SNN ratio that minimize the weighted loss function $\mathrm{L}_{\mathrm{overall}}(r)$. Ideally, one could apply optimization methods that use the gradient information to optimize the SNN ratio. However, obtaining the analytical expression of $\mathrm{L}_{\mathrm{P}}(r)$ is challenging, especially when incorporating Hessian-based optimization techniques, which complicates the mapping relationship. To this end, we employ the subgradient method to approximate the derivatives of $\mathrm{L}_{\mathrm{overall}}(r)$ and carry out the optimization. The subgradient method allows us to estimate the gradients of the weighted loss function, while the finite difference method is used to approximate the first derivatives of $\mathrm{L}_{\mathrm{P}}(r)$ during the iterative process. Specifically, the gradient approximation can be calculated by

$$\frac{\partial \mathrm{L}_{\mathrm{P}}(r)}{\partial r} \leftarrow \frac{\mathrm{L}_{\mathrm{P}}(r+\delta r) - \mathrm{L}_{\mathrm{P}}(r)}{\delta r} \tag{5}$$

where δr represents the disturbance applied to r. In our experiments, we used $\delta r = 0.01$. This combination of methods facilitates finding the optimal or quasi-optimal solution for the SNN ratio.

To demonstrate the feasibility of the optimization process, we consider equivalent weighted coefficients (i.e., $\lambda_1 = \lambda_2 = 1$) and apply the proposed method to the S-MNIST task. Supplementary Figure 7 shows three examples of the optimization process with random initial points. It is observed that the optimization processes can quickly converge to quasi-optimal SNN ratios within the optimization framework no matter where the initial ratio is located, demonstrating the effectiveness of the proposed optimization method. Our approach provides an easy-to-use tool for users to determine the SNN ratio in their applications.



Supplementary Figure 7. Automatic search for optimizing the SNN ratio. The SNN ratio is optimized by minimizing a hybrid loss function, which consists of the normalized accuracy loss L_P and the computational cost loss L_{cost}, through subgradient methods. (a)-(c) illustrate the search process using three examples with random initial SNN ratios.

References

1. G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," *Advances in neural information processing systems*, vol. 31, 2018.
2. A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Unsupervised event-based learning of optical flow, depth, and egomotion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 989–997.
3. Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, "Direct training for spiking neural networks: Faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1311–1318.
4. W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 2661–2671.
5. J. Hagenaars, F. Paredes-Vallés, and G. De Croon, "Self-supervised learning of event-based optical flow with spiking neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 7167–7179, 2021.
6. G. Gallego, M. Gehrig, and D. Scaramuzza, "Focus is all you need: Loss functions for event-based vision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 280–12 289.
7. C. Scheerlinck, H. Rebecq, D. Gehrig, N. Barnes, R. Mahony, and D. Scaramuzza, "Fast image reconstruction with an event camera," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 156–163.
8. P. Charbonnier, L. Blanc-Feraud, G. Aubert, and M. Barlaud, "Two deterministic half-quadratic regularization algorithms for computed imaging," in *Proceedings of 1st international conference on image processing*, vol. 2. IEEE, 1994, pp. 168–172.