nature communications

Article https://doi.org/10.1038/s41467-024-51641-x

Adaptive spatiotemporal neural networks through complementary hybridization

Received: 12 October 2023

Accepted: 12 August 2024

Published online: 27 August 2024

Check for updates

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

Processing spatiotemporal data sources with both high spatial dimension and rich temporal information is a ubiquitous need in machine intelligence. Recurrent neural networks in the machine learning domain and bio-inspired spiking neural networks in the neuromorphic computing domain are two promising candidate models for dealing with spatiotemporal data via extrinsic dynamics and intrinsic dynamics, respectively. Nevertheless, these networks have disparate modeling paradigms, which leads to different performance results, making it hard for them to cover diverse data sources and performance requirements in practice. Constructing a unified modeling framework that can effectively and adaptively process variable spatiotemporal data in different situations remains quite challenging. In this work, we propose hybrid spatiotemporal neural networks created by combining the recurrent neural networks and spiking neural networks under a unified surrogate gradient learning framework and a Hessian-aware neuron selection method. By flexibly tuning the ratio between two types of neurons, the hybrid model demonstrates better adaptive ability in balancing different performance metrics, including accuracy, robustness, and efficiency on several typical benchmarks, and generally outperforms conventional single-paradigm recurrent neural networks and spiking neural networks. Furthermore, we evidence the great potential of the proposed network with a robotic task in varying environments. With our proof of concept, the proposed hybrid model provides a generic modeling route to process spatiotemporal data sources in the open world.

It is not an exaggeration to say that the hungry demand for data learning is at the core of the ongoing age of intelligence. Being able to effectively process multi-scale complex spatiotemporal information is important for many real-world applications, such as handling video in self-driving cars, interpreting written text in mobile reading apps, and managing various types of sensor data in outdoor robots. However, it remains a significant challenge to process such complex data accurately, reliably, and efficiently, particularly in varying environments with different performance requirements.

In mainstream machine learning, non-spiking recurrent neural networks (RNNs) serve as a pivotal model for processing spatiotemporal data. Unlike traditional feedforward architectures, RNNs incorporate recurrent connections into standard artificial neural network (ANN) models, enabling them to capture temporal patterns. While RNNs are extensively employed in diverse applications like

[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, China. [3]Institute of Theoretical Computer Science, Graz University of Technology, Graz, Austria. [4]School of Computer Science, Peking University, Beijing, China. [5]Center for Energy-Efficient Computing and Applications, Peking University, Beijing, China. [6]These authors contributed equally: Yujie Wu, Bizhao Shi. e-mail: leideng@mail.tsinghua.edu.cn

speech recognition[1], language modeling[2], and state control[3], they cannot learn long temporal dependency due to the gradient vanishing during backpropagation (BP) learning[4]. To address this issue, variants such as long short-term memory (LSTM) networks[5] have been developed. These advanced models, equipped with additional gated units, excel at capturing long-term temporal dependencies but come at the cost of increased computational complexity.

Concurrently, there is growing interest in neuromorphic computing as an alternative pathway for developing intelligent models that are both computationally efficient and biologically inspired. Spiking neural networks (SNNs), regarded as the third generation of neural networks[6], are the most famous family of neuromorphic models. The behaviors of each spiking neuron are described by the nonlinear dynamics of the membrane potential and the binary spiking mechanism for the communication between adjacent neurons[7]. Distinct from the extrinsic dynamics of RNNs induced by external recurrence, the dynamics in SNNs intrinsically exist within each neuron. The intra-neuron temporal dynamics and the spatial dataflow through the network make SNNs well-suited for processing spatiotemporal data. Up to now, SNNs have been extensively used for spike stream processing[8], speech recognition[9], ECG signal analysis[10], state control[11], and so forth.

Machine-learning-oriented RNNs use intense matrix multiplications for computation and continuous activations for inter-neuron communication; on the contrary, neuromorphic-computing-oriented SNNs use sparse matrix accumulations for computation and binary spikes for inter-neuron communication. Compared to the continuous activation state space of RNNs, the spike states of SNNs usually evolve in a discrete space. With these distinctions, RNNs have been evidenced to achieve higher accuracy on conventional continuous data sources (e.g., speech signals and language texts) while SNNs are more suited for discrete data sources[12] such as the event stream collected by dynamic vision sensors (DVS)[13]. Owing to the natural filtering effect of the membrane potential leakage along with the spike firing and reset mechanisms of spiking neurons, SNNs have demonstrated strong robustness against variations in temporal resolution[12] and adversarial attack[14]. In addition, owing to the binary format of spikes and the sparsity of spiking activities, the computational cost of an SNN model can be much lower than its non-spiking counterpart under the same network structure[12,15].

Based on the above analyses, it can be seen that RNNs and SNNs present different performance results due to the disparate modeling paradigms. However, in practical scenarios, the type of data sources varies, e.g., continuous data or discrete data, and the performance requirement may also be highly diverse. For example, the high functional accuracy attracts the most attention from cloud users, while the low computational cost is more important for energy-restricted edge devices. Furthermore, for many core components in a system, how to guarantee high robustness against internal noise or external attack becomes the primary design consideration. Even though we can build a specific model to accomplish each task, it would be inefficient because researchers cannot directly apply the experiences accumulated in the modeling exploration when the task changes. To escape the one-task-one-model dilemma, a unified modeling framework to realize adaptive accuracy, robustness, and efficiency is highly expected for processing spatiotemporal data in various scenarios.

Here we report a unified modeling framework that creates hybrid spatiotemporal neural networks (HSTNNs) by synergistically combining RNNs and SNNs for processing spatiotemporal data sources. To make the hybrid model learnable, our work builds on a unified learning methodology, backpropagation through time (BPTT) augmented with a surrogate function, which works for both RNNs and SNNs and thus opening the possibility for hybridization. Furthermore, we exploit a classical pruning method[16,17] to realize neuron selection from RNN and SNN populations and further develop a neuron-aware three-stage hybridization solution to create HSTNNs. It leverages the Hessian gradient information and enables automatic learning of a hybrid structure during the training phase. On several typical spatiotemporal dataset benchmarks, HSTNNs demonstrate better adaptive ability in balancing different performance metrics in terms of accuracy, robustness, and efficiency by tuning the configuration between two types of neurons, and usually outperform conventional single-paradigm RNNs and SNNs. With a robotic place recognition task, we evidence the great potential of HSTNNs in varying environments. Overall, the proposed HSTNNs provide an attractive way to adaptively process variable spatiotemporal data sources in the open world.

Results

Creating HSTNNs

Generally, RNNs and SNNs adopt different strategies in neural coding, computation, and communication, leading to varying performance and application suitability on specialized devices. How to incorporate the distinct features of RNNs and SNNs and integrate their complementary advantages is an open but foremost issue for designing HSTNNs. Recent progress in the neuromorphic field has seen a surge of interest in a hybrid approach that converts non-recurrent neural networks into spiking networks[18–22]. Several studies[23,24] have explored integrating non-recurrent ANNs and SNN modules at the layer level. However, elaborating on specifying fixed heterogeneous networks in advance for each specific task is required, and a hybrid approach supporting effectively integrating diverse temporal dynamics and handling spatio-temporal data flows is still lacking.

To maintain the features of different neurons, we take a decoupling strategy to preserve the diverse spatiotemporal dynamics of different neurons and allow hybrid information transmission at the neuron level. The general structure of HSTNNs is shown in Fig. 1a. Each hybrid layer contains two neuron populations, an RNN one with artificial neurons and an SNN one with spiking neurons. Both populations receive the same mixed inputs from the previous layer, independently update the respective spatiotemporal dynamics, and synergistically send mixed outputs by concatenating both RNN and SNN outputs. Within each hybrid layer, each neuron only connects to the neurons belonging to the same population.

To generate optimal dynamics between spiking neurons and non-spiking artificial neurons, we expect the hybrid network can be automatically learned. RNNs are usually trained with the classical BPTT algorithm, while SNNs are widely trained with bio-plausible synaptic plasticity rules, e.g., spike timing dependent plasticity (STDP)[25,26], which is incompatible with BPTT. Fortunately, BPTT has been adapted to SNNs recently by addressing the training convergence problems and the issue of the non-differentiable spiking activities. This progress in the SNN domain lays the foundation for hybridizing RNNs and SNNs under a unified learning framework. Based on BPTT, we propose a three-stage hybrid learning methodology to create HSTNNs, which will be detailed in the following sections.

The core idea of the learning methodology is an evolution-inspired strategy, which combines a unified BPTT learning algorithm and a neuron-aware selection mechanism to select prominent neurons from two redundant neuron populations for building a hybrid network. As shown in Fig. 1b, to explore the optimal structure of the hybrid network, we generate two types of neuron populations in each layer, which represent a redundant number of candidate neurons for constructing the HSTNN. Each population is independent of the other within the layer and the outputs are combined before injecting into the next layer. We adopt BPTT with the surrogate gradient technique to pretrain this initial network to a good point with inter-population interactions. Next, we develop a neuron-aware selection mechanism to measure the importance of different types of neurons, which guides the selection of prominent neurons from the two redundant populations by invalidating unimportant neurons and their connections.

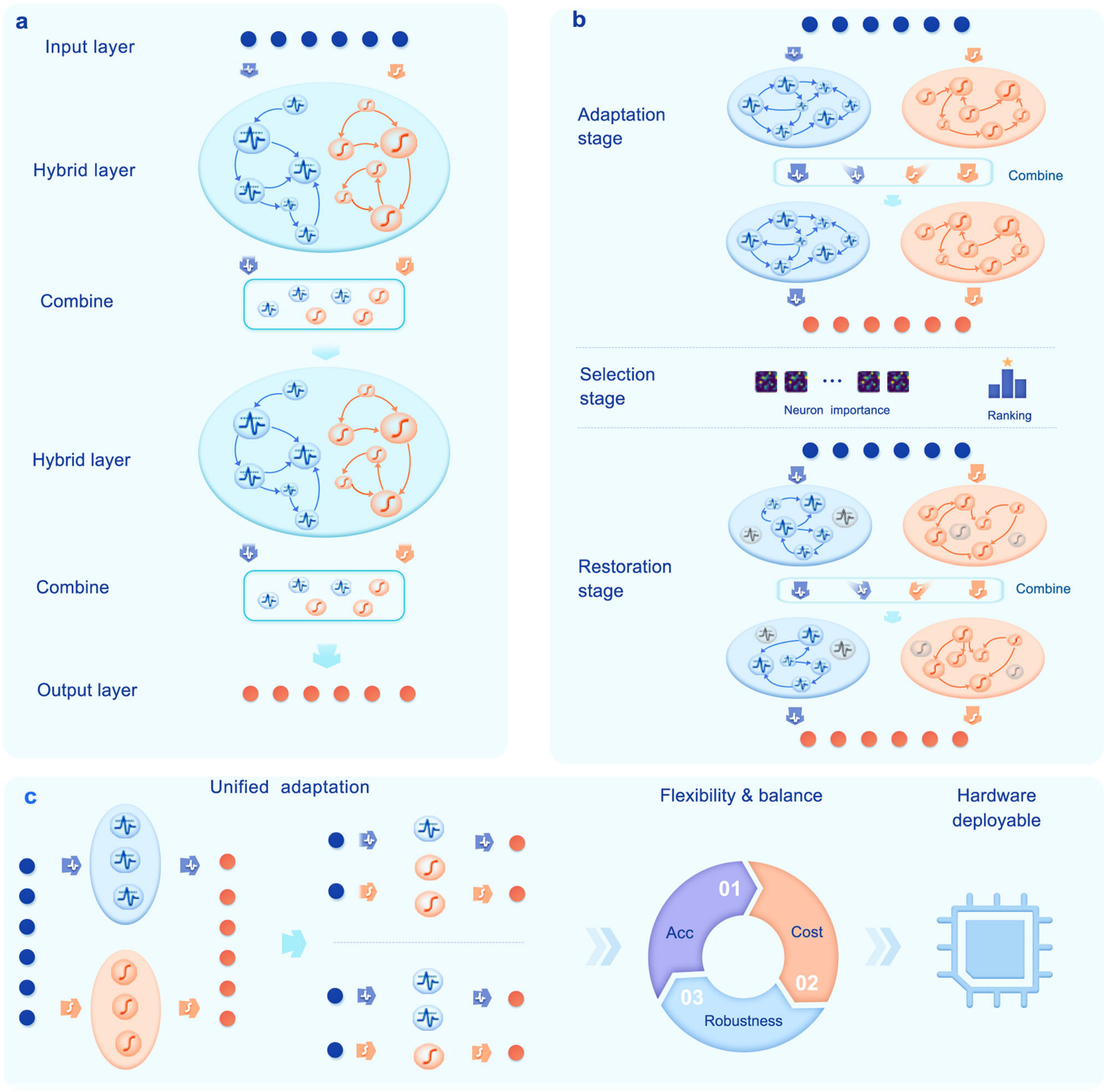


Fig. 1 | Methodology of creating HSTNNs. a The HSTNN architecture. Each hidden layer of an HSTNN contains two types of neurons and their outputs are combined before being injected into the next layer. **b** The three-stage learning process for creating an HSTNN. In the *Adaptation* stage, we create two redundant populations of neurons in each hidden layer and apply the unified BPTT learning algorithm to warm the connection weights. In the *Selection* stage, we propose a neuron-aware selection mechanism to measure the importance of neurons and select important neurons. In the *Restoration* strategy, we shrink the network by invalidating unselected neurons and their connections and retrain the compact model until convergence. **c** HSTNNs exhibit several advantageous features: an HSTNN can be conveniently initialized under a unified *Adaptation* stage no matter what the specific task scenario is; it enables a flexible balance between accuracy, cost, and robustness to satisfy variable performance requirements in practice; it can be deployed on neuromorphic hardware for constructing an efficient application system.

Finally, we retrain the resulting compact hybrid network after neuron selection until convergence. Notice that the ratio between two types of neurons is given at the beginning, while which neurons to select is automatically determined during learning.

The proposed HSTNNs demonstrate several advantageous features benefiting practical usages, as depicted in Fig. 1(c). First, the proposed hybridization approach adopts a unified adaptation strategy to create redundant neuron populations no matter what the target task is, thereby easing its usage. Second, HSTNNs enable a flexible balance between accuracy, cost, and robustness according to the actual need by customizing the ratio of different types of neurons, which is able to satisfy variable performance requirements in practice. Third, HSTNNs exhibit better accuracy and robustness compared to single-paradigm networks with appropriate neuron configurations. Last, although HSTNNs represent a novel paradigm, they can be deployed on existing neuromorphic hardware, especially on the ones with hybrid computing architectures[11,27,28], which promises the construction of efficient application systems.

Three-stage hybrid learning

HSTNNs use a three-stage learning methodology to generate the hybrid network, including *Adaptation*, *Selection*, and *Restoration* stages. The three stages gradually extract a compact hybrid network from an initialized redundant network in an evolutionary manner.

In the *Adaptation* stage, an HSTNN first expands redundant neuron pools with different types of neurons at each hidden layer. The neurons in each pool are governed by their respective spatiotemporal dynamics and the output representations are merged and then propagated to the next layer. The surrogate gradient technique[12,29] is used to handle the derivative of the non-differentiable spiking function, and the BPTT learning algorithm is then applied to warm the connection weights.

The *Selection* stage aims at identifying critical neurons and pruning the unimportant neurons to satisfy a target neuron configuration. To quantitatively describe the neuron configuration, in the following, we introduce the SNN ratio as the ratio of the number of spiking neurons to the total number of hidden neurons after the three-stage learning. To produce an effective selection mechanism, we get inspiration from the OBS method[16], a classical network pruning method for shallow feedforward networks, and extend it for the selection of important neurons in hybrid recurrent networks. Particularly, we use the second-order gradient information of synaptic connections around the local minimum of the loss function as the basic measurement for importance, collect the importance scores of different afferent connections for each neuron, and finally evaluate the neuron importance according to the accumulated gradients spanning the temporal domain. Because of the different types of gradient information of spiking and non-spiking neurons, we introduce a grouping selection mechanism that ranks the same type of neurons across all layers and constrains the overall number of neurons according to the specific SNN ratio. By doing so, the *Selection* stage invalidates unimportant neurons and shrinks the redundant structures to an expected compact level.

Since the selection process is independent of the adaptation stage, this learnable architecture is quite efficient and flexible to meet the actual need without the guidance of expertise to tailor a network architecture for each specific task. Given the compact network structure, the *Restoration* stage finally fine-tunes the remaining neural interactions to get the final HSTNN.

The three-stage learning methodology employs the surrogate gradient technique to approximate the first- and second-order gradient information of the non-differentiable spiking activities, which potentially influences the learned interaction between spiking and non-spiking activities. Our empirical analysis is shown in Supplementary Fig. 2 that the specific format of the surrogate function proves to have little impact on the learning performance of the HSTNN. HSTNNs can learn a similar profile of neuron importance under different surrogate functions (see Supplementary Fig. 2b, c), leading to a competitive learning performance at the end of the third stage (see Supplementary Fig. 2d, e). This indicates that the proposed hybridization approach demonstrates stability against various hyperparameters of the surrogate functions during the learning process.

Comprehensive performance in terms of task performance and the computational cost

The HSTNN presents a general hybrid strategy for integrating different network paradigms, suitable for various sequential learning tasks. We first evaluate its comprehensive performance in terms of both task accuracy and the computational cost in four different types of tasks, as shown in Fig. 2. For tasks on PTB, S-MNIST, and N-MNIST datasets, HSTNNs with two hybrid fully-connected hidden layers are implemented. For the more challenging DVS-Gesture dataset, a nine-layer convolutional-based network structure is employed (see Methods).

As depicted in Fig. 2a–d, single-paradigm networks, SNNs, and RNNs, exhibit variable performance on different datasets: RNNs perform better on traditional deep-learning-oriented datasets like S-MNIST and PTB datasets, while SNNs excel on neuromorphic-computing-oriented datasets like N-MNIST and DVS-Gesture. This prominent difference may result from the radical difference in neural coding and computation properties of the two networks. Compared to SNNs, the high-precision neural representation of RNNs is advantageous in handling continuous-value-based text analysis tasks like PTB. Conversely, the natural filtering effect of the membrane potential leakage and the spike rate coding schemes may enhance the robustness of SNNs against input fluctuations, leading to better performance of SNNs on neuromorphic datasets, as evidenced by the results on N-MNIST and DVS-Gesture.

Combining Fig. 2a–h can further derive three key conclusions about HSTNNs. First, HSTNNs with suitable neuron configurations can outperform both single-paradigm SNNs and RNNs. This improvement is likely due to the hybrid information representation of non-spiking and spiking neural networks with richer neuronal computation mechanisms, which increases learning nonlinearity and integrates the complementary strengths of SNNs and RNNs in addressing specific tasks. For instance, such integration can increase the representation precision of single-paradigm SNNs on the non-spiking PTB dataset and enhance the robustness of single-paradigm RNNs against input fluctuations on the spiking N-MNIST dataset.

Second, Fig. 2e–h shows that HSTNNs demonstrate adaptive balance in accuracy and the computational cost between SNNs and RNNs. The computational cost gradually decreases as the SNN ratio grows owing to the high efficiency of SNNs dominated by sparse accumulation operations. In Fig. 2i, we further take the N-MNIST dataset as an example to evaluate the correlation between accuracy and computational cost. The lighter color of a bubble indicates a larger SNN ratio with a lower computational cost. Results show that HSTNNs are able to produce a better comprehensive performance solution with higher accuracy and a lower computational cost, demonstrating the effectiveness of the proposed HSTNN model.

Third, HSTNNs provide an effective hybridization method compared with the direct hybridization method. To evidence the superiority of our three-stage hybridization, we build a baseline model for comparison. This baseline model, named the directly-hybrid model, trains a hybrid model from scratch under a given SNN ratio without employing the three-stage learning process. As shown in Fig. 2a–d, HSTNNs perform better than the directly-hybrid models. Furthermore, as observed in Fig. 2j, the improvement of HSTNNs over directly-hybrid models becomes more pronounced when constructing smaller-size hybrid neural networks, a practical constraint commonly considered in real-world edge systems with limited resources. These comparisons reflect the effectiveness of the elaborate three-stage learning methodology: expanding the representation dimension initially and then meticulously selecting important neurons from the hybrid redundant populations, rather than directly training a compact hybrid network from scratch.

Robustness analysis against noise, frame-loss, and adversarial attack

We next examine the robustness of sole SNNs, sole RNNs, and HSTNNs on the deep-learning-oriented S-MNIST and the neuromorphic-computing-oriented N-MNIST dataset. We evaluate the model robustness in three aspects including random noise robustness, frame-loss robustness, and adversarial attack robustness. For the random noise robustness, considering the diverse data characteristics of different datasets, we add the Gaussian noise into the testing samples of S-MNIST and the salt-and-pepper noise into the testing samples of N-MNIST. For the frame-loss robustness, we randomly mask some sequence information of each frame of the testing sample. For the adversarial attack robustness, we generate untargeted adversarial samples with a small level of perturbation on the raw samples (see Methods). Three selected digits are shown in Fig. 3a to illustrate different formats of testing samples in robustness experiments. All models are trained on standard training sets while evaluated on preprocessed testing sets with testing samples described above. We record the performance of HSTNNs compared to that of sole RNNs and

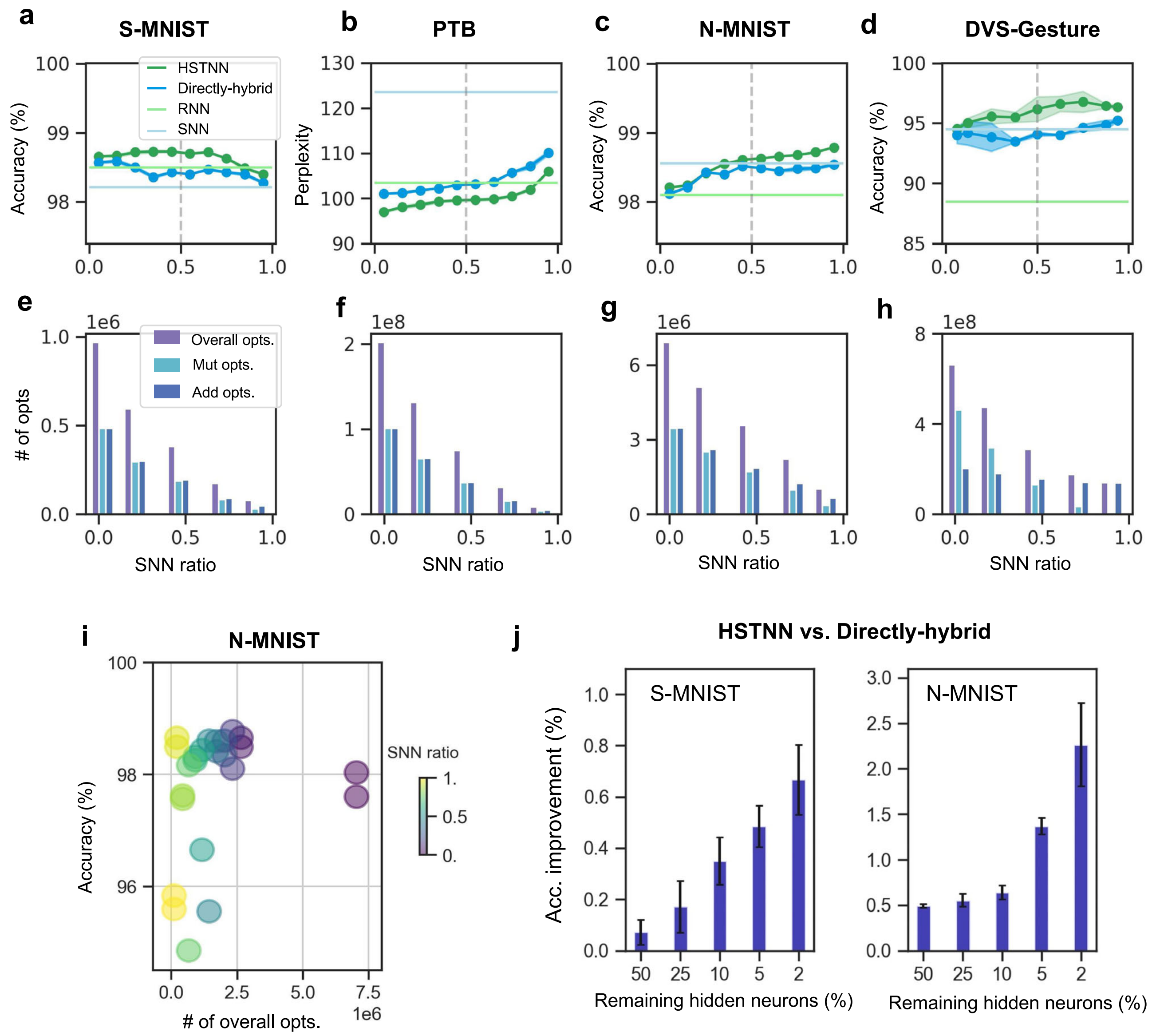


Fig. 2 | Comprehensive evaluation of HSTNNs in terms of task performance and the computational cost. Impact of the SNN ratio on accuracy (upper) for (**a**) S-MNIST, (**b**) PTB, (**c**) N-MNIST, and (**d**) DVS-Gesture datasets. Impact of the SNN ratio on the number of operations (lower) for (**e**) S-MNIST, (**f**) PTB, (**g**) N-MNIST, and (**h**) DVS-Gesture datasets. A two-hidden-layer network structure is adopted for PTB, S-MNIST, and N-MNIST, and a convolutional structure for DVS-Gesture (see Methods). Note that task performance on the PTB dataset is measured by perplexity (ppl), where lower is better. S-MNIST is a variant of the standard MNIST dataset in which the images are input into the model sequentially, one column at a time. **i** Comprehensive analysis of the trade-off between accuracy and the computational cost for the N-MNIST dataset, where the computational cost is measured by the total number of multiplication and addition operations. **j** Analysis of average accuracy improvement by HSTNNs compared to directly hybrid models, measured by the ratio of remaining hidden neurons in the *Restoration* stage to those in the *Adaptation* stage. All error bars represent the standard deviation over five repeated trials.

SNNs in the three types of noise tests in Fig. 3b. Note that for the noise and frame-loss robustness experiments, higher recognition accuracy indicates better robustness, while for the adversarial attack robustness experiments, a lower attack success rate implies better resistance to adversarial attacks.

We observe two prominent phenomenons in Fig. 3b–c. First, as presented in Fig. 3b, in both noise robustness and frame loss robustness experiments, HSTNNs perform comparably to the best single-paradigm models, i.e., RNNs or SNNs. In the adversarial attack robustness experiment, HSTNNs even demonstrate better robustness than both RNNs and SNNs. Second, we observe in Fig. 3c that RNNs and SNNs exhibit different robustness performances in noise and frame loss experiments. As the noise level increases, measured by the standard variance of the Gaussian noise or the frame loss probability on S-MNIST, it is noted that HSTNNs perform closer to the single-paradigm models with stronger robustness in different robustness tests and achieve significantly higher accuracy than the other single-paradigm networks. This implies this hybridization enables to inherit the robustness advantages of single-paradigms of models.

Scalability of HSTNNs in integrating different neuron modules and architectures

The proposed neuron-wise hybridization approach facilitates the easy incorporation of more complex neuronal models and deeper network structures. We have demonstrated in Fig. 2 that the proposed hybrid approach enables applying to different shallow network structures in solving various sequential learning. Now we further quantitatively analyze the scalability of HSTNNs in integrating different neuronal models, including vanilla RNN (vRNN), LSTM, LIF, and adaptive LIF (ALIF), and applying the hybrid approach to deeper network structures.

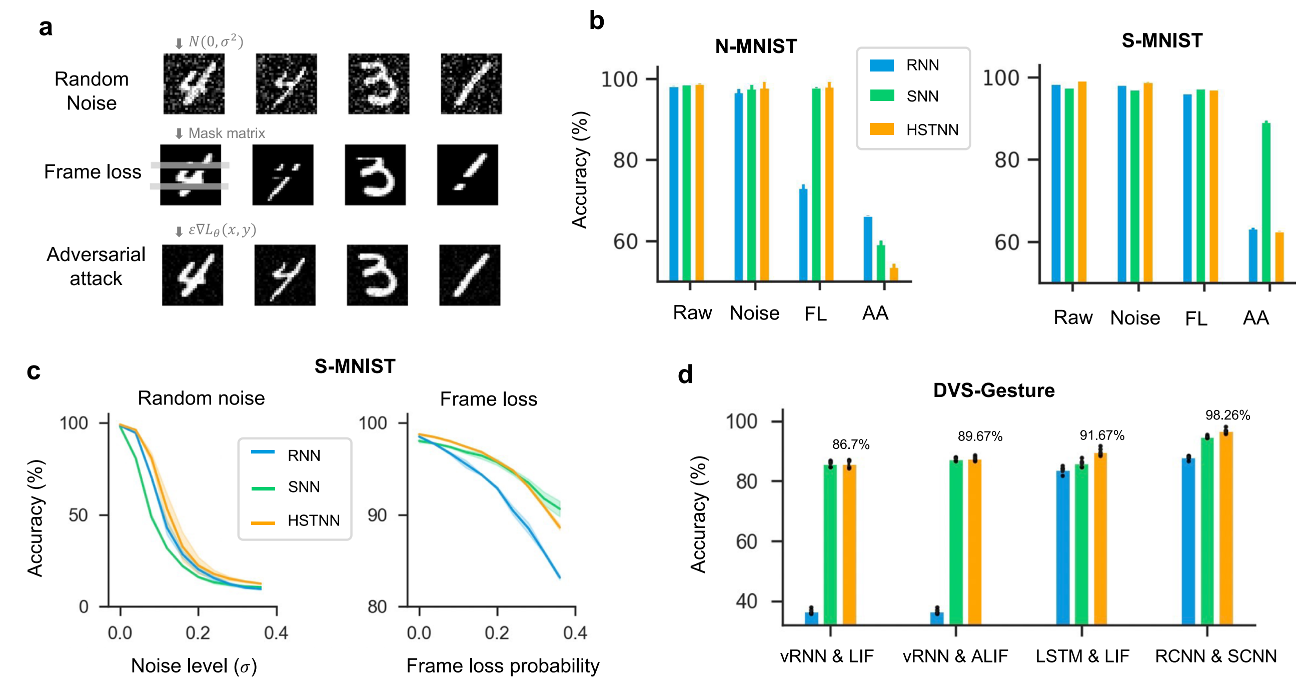


Fig. 3 | Validating the robustness and scalability of HSTNNs. a Illustration of three-digit samples in varied formats for different robustness experiments. Following the training phase, we introduce distinct noise types into the testing samples and report the average accuracy across the entire testing dataset. **b** Comparison of noise robustness, frame-loss (FL) robustness, and adversarial attack (AA) robustness between HSTNNs under the optimal ratios and single-paradigm networks on S-MNIST and N-MNIST datasets. **c** Analysis of the robustness with increasing noise levels (left) and frame loss probability (right). It is observed that RNNs and SNNs exhibit distinct robustness advantages for the two types of noises, while HSTNNs demonstrate a complementary robustness profile. **d** HSTNNs are scalable for integrating neuron models with various types of temporal dynamics, evidenced by accuracy improvements when incorporating more complex neuronal computation features. The HSTNNs with the optimal SNN ratios are reported here for comparison. Error bars represent the standard deviation, and the numbers above the yellow bars denote the best accuracies of HSTNNs over five runs.

Table 1 | Comparison with advanced models on neuromorphic datasets

Model	Dataset	Network Structure	Best Acc. (%)	Mean ± SD (%)
SNN model[8]	N-MNIST	5-layer SCNN	99.35	N/A
SNN model[44]		7-layer SCNN	99.40	N/A
RNN model (our work)		6-layer RCNN	99.55	99.51 ± 0.04
SNN model (our work)		6-layer SCNN	99.56	99.54 ± 0.02
Hybrid model (our work)		6-layer hybrid CNN	99.63	99.61 ± 0.02
SNN model[45]	DVS-Gesture	14-layer SCNN	97.22	N/A
SNN model[46]		14-layer SCNN	97.57	N/A
RNN model (our work)		9-layer RCNN	88.54	87.73 ± 0.87
SNN model (our work)		9-layer SCNN	95.49	94.90 ± 0.53
Hybrid model (our work)		9-layer hybrid CNN	98.26	97.11 ± 1.06
SNN model[47]	CIFAR10-DVS	9-layer SCNN	N/A	67.22 ± 0.43
SNN model[48]		ResNet-19	67.80	N/A
SNN model[46]		VGG11	74.80	N/A
RNN model (our work)		13-layer RCNN	72.50	72.30 ± 0.20
SNN model (our work)		13-layer SCNN	77.00	76.10 ± 0.79
Hybrid model (our work)		13-layer hybrid CNN	78.40	77.47 ± 0.81

Best accuracy, mean, and standard deviations were computed for each single-paradigm model and our hybrid model with the optimal SNN ratio over three trials. Source data are provided as a Source Data file.

Figure 3d shows the results of building HSTNNs with different neuronal models on DVS-Gesture. When comparing the second to fourth groups of bars with the first group, it becomes evident that including more complex neuronal dynamics enhances the overall task performance of single-paradigm spiking and non-spiking neural networks. Notably, one can see that the improvements of HSTNNs for the third and fourth groups are more prominent compared to those for the first and second groups, where the spike-based models demonstrate overwhelmingly better performance. This suggests that the effectiveness of HSTNNs in exploring a complementary and superior solution depends on the original performance difference between single-paradigm networks. Namely, in the case where two single-paradigm networks perform comparably, the HSTNN has a larger chance to produce a better hybridization solution. In addition to DVS-Gesture, we also test the scalability of the proposed approach in deep convolutional network structures (see Methods) on N-MNIST and

CIFAR10-DVS. The results, displayed in Table 1, show that integrating convolutional neuron models can consistently enhance the performance of single-paradigm networks and surpass other advanced models. This underscores the potential of combining heterogeneous neuron types within a single hybrid network to improve task performance.

Adaptability of HSTNNs in varying environments

In addition to evaluating HSTNNs on standard datasets, we further assess their adaptability with a robotic place recognition task in real-world varying environments. To conduct the experiments, we utilize a robot platform developed by our previous study[30], as depicted in Fig. 4a, where the robot navigates in different environments. The objective of this task is to accurately recognize the specific place based on event-based and frame-based vision inputs, as illustrated in Fig. 4b. The entire path is divided into 100 classes representing distinct places, and we collect event-based and frame-based data using a DVS camera and an RGB-D camera from three different environments: (i) an indoor environment with adequate lighting condition (env1), (ii) an outdoor environment with varying lighting condition (env2), and (iii) an indoor environment with low lighting condition (env3).

The results obtained from these place recognition experiments highlight the flexible adaptability of HSTNNs to varying environmental conditions. Figure 4c–g presents a comparative analysis of HSTNNs with varying SNN ratios, including single-paradigm RNNs and SNNs across the three environments. The results indicate that the optimal recognition accuracy varies across different SNN ratios and demonstrate the suitability of different hybrid paradigms for specific environmental conditions. This adaptability enables HSTNNs to perform effectively in variable scenarios. Figure 4h further provides a comprehensive comparison between HSTNNs and single-paradigm RNN/SNN models in terms of accuracy and computational cost. Notably, HSTNNs outperform the single-paradigm models, demonstrating the superiority of the hybrid modeling paradigm. Figure 4h also highlights the trade-off between accuracy and the computational cost, as the best accuracy achieved by HSTNNs corresponds to different numbers of operations. This flexibility empowers practical users to make informed decisions based on their specific requirements.

The performance adaptability, robustness to varying environments, and consideration of the computational cost make HSTNNs a promising approach for processing spatiotemporal data sources in the open world. The findings from the above robotic place recognition experiments evidence the great potential of HSTNNs in achieving improved accuracy compared to single-paradigm models and providing a flexible solution for different application environments.

Hardware deployability of HSTNNs

Applying neural network models in practice depends on efficient hardware. In current intelligent machines, general-purpose processors such as GPUs are the mainstream platforms for running neural networks. Although they can perform ANNs efficiently, there exists a big performance gap when executing SNNs. Neuromorphic processors are another family of hardware that can perform SNNs efficiently but support RNNs inadequately. Therefore, our HSTNNs with hybrid computation of both RNNs and SNNs are not suited for these single-paradigm-oriented hardware platforms. Fortunately, hybrid-paradigm neuromorphic chips[31–33] emerged in recent years, which show promising performance no matter running RNNs or SNNs. To validate the application potential of HSTNNs, we select a recent hybrid neuromorphic chip, TianjicX[34], as the platform for execution efficiency analysis.

We have implemented RNNs, SNNs, and HSTNNs with different SNN ratios on the TianjicX chip. All the networks contain three layers running in a pipeline on the chip as illustrated in Fig. 5a. We employed two mapping strategies, namely fixed-core mapping and variable-core mapping, for different hybrid layers, as presented in Fig. 5b. The small layers are mapped to two fixed cores respectively used for computing SNN and RNN modules, i.e., the fixed-core mapping strategy, while the larger layers are mapped to more cores where the computational resources for SNN and RNN modules are proportional to the number of neurons, i.e., the variable-core mapping strategy. The choice of the strategy depends on the potential for parallel execution of the layer across multiple cores, which will be detailed in Methods.

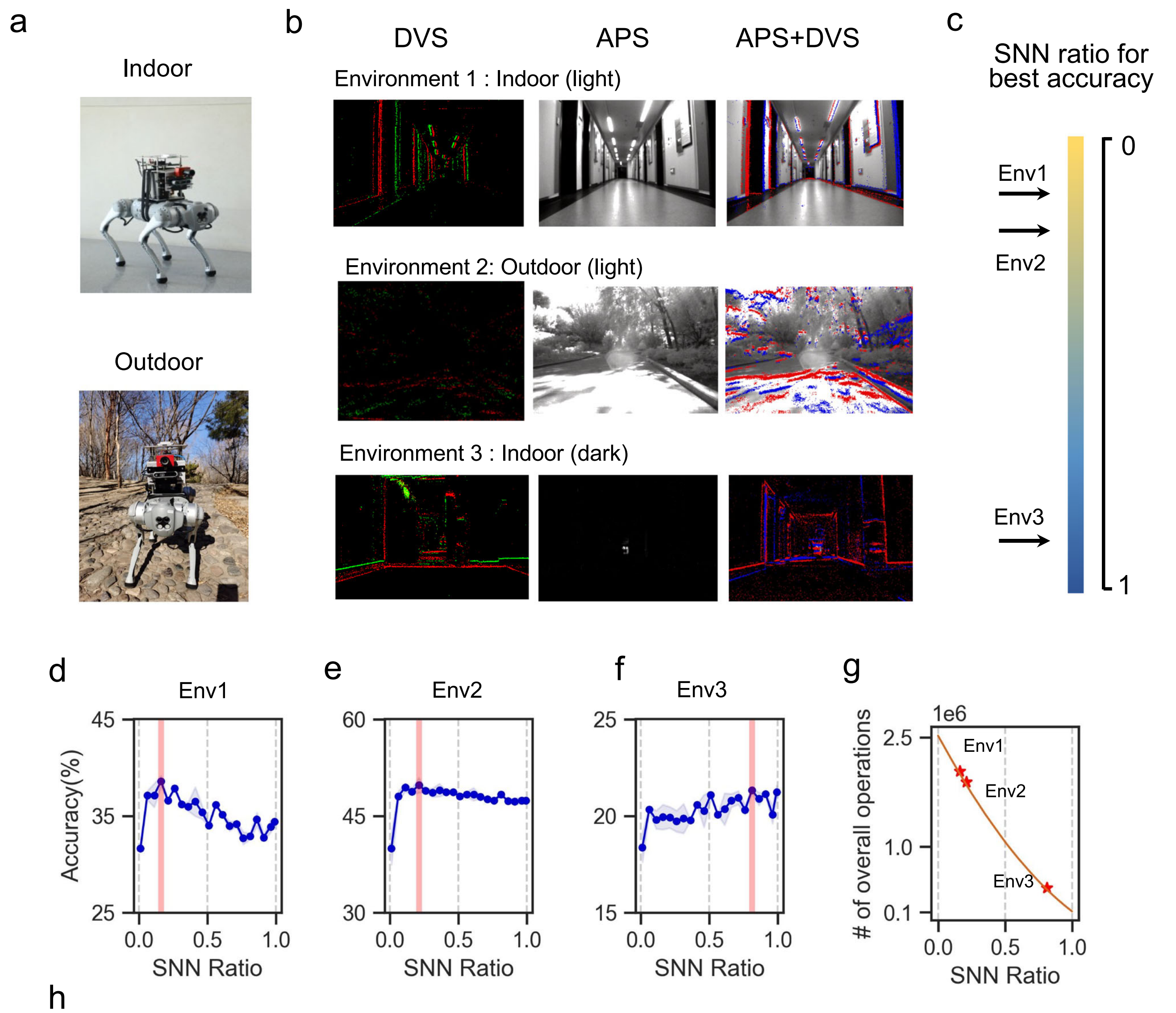
To minimize the effect of the mismatch between the network structure and the hardware architecture, we add extra restrictions to HSTNNs. First, the input size of neuron populations is set to multiples of 16 while the output size is set to multiples of 32 so that the networks can exploit parallelism within each core. Second, we fix the total number of neurons in each layer for HSTNNs with different SNN ratios. This restriction solves the problem that the *Selection* stage with only a global constraint on the entire network might generate a variable number of neurons in each layer, which makes the hardware mapping unfriendly and causes unfairness in comparing execution performance.

Figure 5c shows that, in general, the higher the ratio of SNN neurons in the network, the shorter the execution latency, as recurrent connections in the RNN consume additional computation. The execution latency of the sole SNN on S-MNIST is significantly higher than that of the HSTNN at the SNN ratio of 0.75 because the sole SNN only utilizes a single SNN core for computation in the fixed-core mapping strategy while the hybrid HSTNN can use both cores. By measuring both the execution latency and dynamic power consumption on the chip, we calculate the dynamic energy consumed by RNNs (SNN ratio 0), SNNs (SNN ratio 1), and HSTNNs for inferring one sample. As given in Fig. 5c, the energy consumption and the SNN ratio of HSTNNs are negatively correlated on both S-MNIST and N-MNIST datasets, regardless of the mapping strategy. These results are consistent with the previous analysis of the computational cost and once again demonstrate the flexibility of HSTNNs. As the SNN ratio increases, the heavy computation of RNNs is reduced, which results in lower energy consumption. More details of the mapping strategies and experimental results are available in Methods and Supplementary Fig. 3.

Discussion

We presented a generic hybridization approach that can maintain and integrate the complementary features of RNNs and SNNs, promising a unified effective way to process different types of spatiotemporal data. We observed that RNNs and SNNs have shown divergent performances across six distinct types of tasks. By leveraging their complementary features through our hybrid models, we demonstrated that the HSTNNs not only surpass single-paradigm models in comprehensive performance but also exhibit superior robustness against noise, frame loss, and adversarial attack. Furthermore, the adaptability of HSTNNs to diverse environmental conditions was evidenced in the robot place recognition task. The flexible hybrid paradigm yielded optimal recognition accuracy in a variety of lighting conditions, indicating its potential for handling the complexity and variability of real-world applications. Even though HSTNNs integrate two types of neurons, they can be deployed on emerging neuromorphic chips with the hybrid architecture for efficient execution.

Interestingly, HSTNNs exhibit intriguing similarities to the coding strategies and integration of continuous and spiking activities observed in the human brain for information processing. The brain, renowned as a hybrid learning system, employs diverse types of neuron populations and a range of coding schemes to tackle complex spatiotemporal tasks. HSTNNs can actually achieve similar functionality by leveraging different coding strategies of SNNs and RNNs. We observed that the interaction between continuous and spike-based neural activities can alter the spiking activities of the spiking neuron



h

Env 1: indoor (light)			Env 2: outdoor (forest)			Env 3: indoor (dark)		
Model	# opts.(10^6)	Acc.(%)	Model	# opts.(10^6)	Acc. (%)	Model	# opts.(10^6)	Acc. (%)
RNN	2.52	31.63	RNN	2.52	39.97	RNN	2.52	18.38
SNN	0.11	34.4	SNN	0.11	47.37	SNN	0.11	21.23
HSTNN	2.04	38.6	HSTNN	1.89	49.80	HSTNN	0.44	21.32

Fig. 4 | Validating the adaptability of HSTNNs with a robotic place recognition task in varying environments. a Experimental setup for data collection. **b** The HSTNNs are tested in three different environments: (i) env1, an indoor environment with adequate lighting conditions; (ii) env2, an outdoor environment with varying lighting conditions; and (iii) env3, an indoor environment with low lighting conditions. The HSTNNs simultaneously receive inputs from a dynamic vision sensor (DVS) device (the first column) and an RGB-D camera (the second column). **c** The best place recognition accuracy in different environments is achieved at different SNN ratios. **d**–**f** Accuracy of the HSTNNs in the three environments, with error bars representing the standard deviation over three trials. **g** The overall number of operations performed by the HSTNNs in this task. **h** Comprehensive comparison of performance results between SNN/RNN models and the HSTNNs with the optimal SNN ratio. Acc.: accuracy and opts.: operations.

population (see Supplementary Figs. 5, 6). This amalgamation of various neuronal dynamics and coding strategies in HSTNNs embraces the diversity and richness of the brain's own computational strategies, an aspect that has been underscored by contemporary neuroscientific research[35,36]. These parallels not only highlight the relevance and potential of HSTNNs for handling diverse real-world applications but also provide hints to understanding the design of more robust and adaptable systems for artificial intelligence.

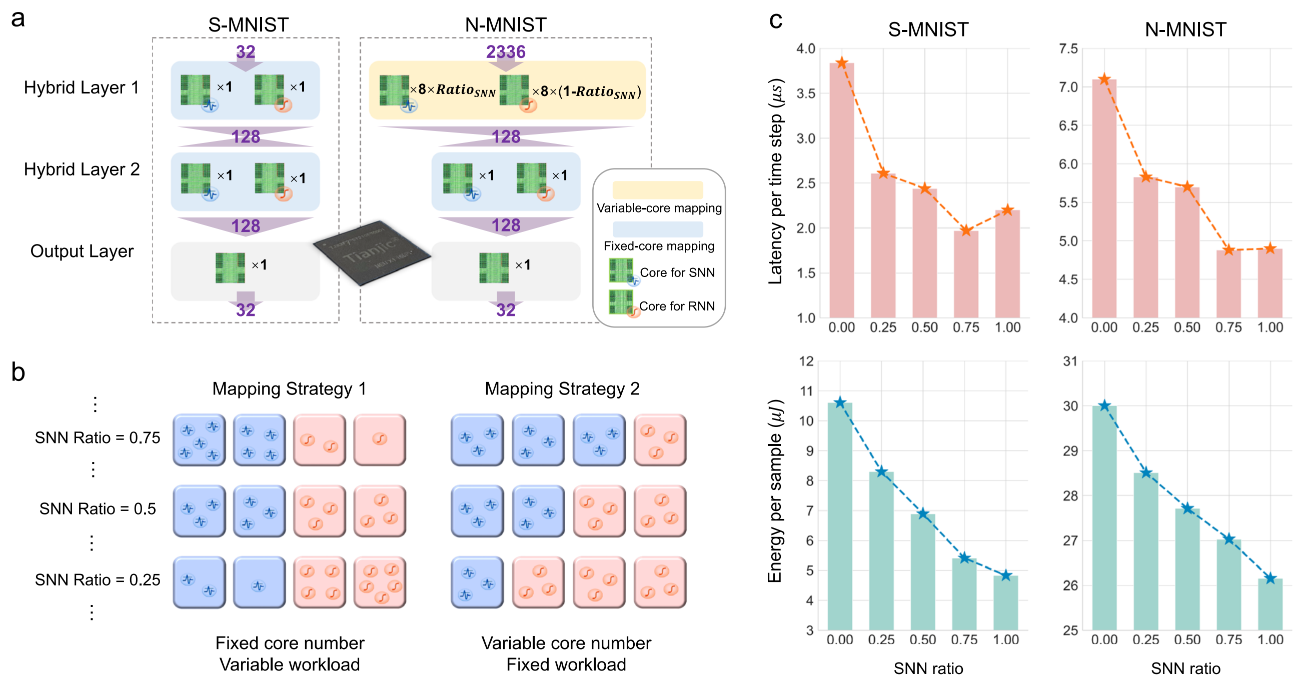


Fig. 5 | Comparing the efficiency of RNNs, SNNs, and HSTNNs on neuromorphic hardware. a Implementation of HSTNNs on the TianjicX chip. Two three-layer neural networks on S-MNIST and N-MNIST datasets are mapped onto the chip with different mapping strategies. **b** Illustration of the two-layer mapping strategies. **c** Execution latency and energy consumption of RNNs, SNNs, and HSTNNs with different SNN ratios.

How to determine an optimal SNN ratio in HSTNNs and thereby achieve the balance between task performance and the computational cost is a crucial but open issue. This optimal SNN ratio is highly context-dependent and varies according to specific user requirements, as the weights assigned to accuracy and the cost differ across different environments. We show a heuristic method in Supplementary Information to address this challenge and provide an optimal SNN ratio automatically searched in specific tasks. By formulating the optimization problem and employing approximation optimization methods such as subgradient descent, we demonstrate in Supplementary Fig. 7 the feasibility of finding the optimal solution of the SNN ratio. It allows users to customize the model based on their specific needs and achieve an optimal trade-off between accuracy and cost. The practical implications of this optimization process are significant, as it facilitates real-world applications of HSTNNs in practical environments for effective and efficient processing of spatiotemporal data.

Hybrid neural network models gain more and more interests from different fields due to the rapid development of neuroscience and the breakthrough of deep learning[23,31]. Several advantages of layer-wise hybridization have been demonstrated in references[23,24]. These layer-wise hybridization approaches focus on integrating non-recurrent ANN and SNN modules using layer-wise strategies, providing efficient solutions for practical applications such as optical flow estimation and high-speed tracking tasks. In contrast, the proposed neuron-wise hybridization approach in this work offers a finer-grained method of hybridization, enabling real-time interaction of the coding and computational features of different types of neurons. Moreover, the neuron selection strategy employed in the *Selection* stage represents a more general solution that encompasses layer-wise hybridization as a specific case. We demonstrate in Supplementary Fig. 4 that the proposed hybridization model can be applied to optical flow estimation, yielding comparable results to those of specially designed single-paradigm networks.

The proposed HSTNN is a very initial effort to bridge dynamic models in machine learning and neuromorphic computing. As aforementioned, HSTNNs have presented great potential in task performance, model robustness, and computational cost, which provides a flexible trade-off to satisfy variable environments and user requirements under a unified modeling and learning framework. The models showcased in this work are relatively simple, offering considerable scope for further enhancement. For instance, HSTNNs could be enhanced with advanced transformer architectures and create deeper and larger models, enabling the processing of more complex spatiotemporal data. Furthermore, intelligent machines equipped with neuromorphic chips can incorporate HSTNNs to process spatiotemporal information collected by various sensors such as cameras, microphones, electroencephalogram electrodes, and so forth. We look forward to inspiring more investigations for taking complementary features and advantages of computer-science-oriented models and neuroscience-oriented models.

Methods

Establishment of HSTNN

The HSTNN contains an input layer, one or multiple hidden hybrid layers, and a readout layer. Each hybrid layer has a population of non-spiking recurrent neurons and a population of spiking neurons. The size of each population can be adaptively changeable during the training process. To facilitate efficient training of hybrid models, non-spiking recurrent neurons are simulated with synchronized timing as spiking neurons. At each time step, neurons in the two populations receive the same mixed inputs from the previous layer and their outputs in the previous time step, and then update the neural dynamics and generate their respective outputs ($\boldsymbol{r}_t$ or $\boldsymbol{s}_t$). These outputs are combined before being forwarded to the next processing layer. The combined output $\boldsymbol{y}_t^n$ of the n-th hybrid layer at the t-th time step is formalized as:

$$\boldsymbol{y}_t^n = \mathrm{Concat}\left(\mathrm{SNN}\left(\boldsymbol{s}_{t-1}^n, \boldsymbol{y}_t^{n-1} \parallel \boldsymbol{\theta}_{SNN}^n\right), \mathrm{RNN}\left(\boldsymbol{r}_{t-1}^n, \boldsymbol{y}_t^{n-1} \parallel \boldsymbol{\theta}_{RNN}^n\right)\right), n = 2, \ldots, N-1, \quad (1)$$

where N denotes the number of layers. A detailed illustration of the propagation of hybrid information in HSTNN is provided in

Supplementary Fig. 1. To support diverse spike decoding schemes, the HSTNN uses a generic parametric decoder $d(\boldsymbol{x}|\boldsymbol{\omega})$ capable of decoding rate-based or timing-based information from the output spike train $\{\boldsymbol{s}_1^N, \boldsymbol{s}_2^N, \ldots, \boldsymbol{s}_t^N\}$ into a vector representation. The parameter $\boldsymbol{\omega}$ represents the weight matrix assigned to the output spike train in the specific decoding scheme. The decoded spiking information is concatenated with the RNN's outputs to produce a final output via a readout weight $\boldsymbol{W}^N$. This process can be formalized as follows:

$$\boldsymbol{y}_t^N = \boldsymbol{W}^N \mathrm{Concat}\left(\boldsymbol{r}_t^N, d\left(\boldsymbol{s}_1^N, \boldsymbol{s}_2^N, \ldots, \boldsymbol{s}_t^N|\boldsymbol{\omega}\right)\right). \tag{2}$$

In our experiments, we instantiate $d(*)$ in a straightforward form: $d_i = \sum_{t'=1}^{t} \omega_{i,t'} \boldsymbol{s}_{i,t'}^N$, where $d_i(*)$ represents the ith component of $d(*)$. In the neuromorphic dataset including N-MNIST, DVS-Gesture, and CIFAR10-DVS, we decode the rate-based information from output spike trains by setting an equal entry for $\omega_{i,t'} = \frac{1}{t}, \forall i, t'$; in the text analysis tasks, we employ the spike timing information at the last time step by setting $\omega_{i,t} = 1$ and $\omega_{i,t'} = 0$ for $t' < t$.

Neuron models for HSTNN

The HSTNN facilitates the integration of various non-spiking RNN modules and spiking modules. In this work, we primarily instantiate two representative RNN modules, including vanilla RNN and LSTM, as well as two SNN modules, LIF and ALIF[37], for constructing the HSTNN. The behaviours of a vanilla RNN module can be described by

$$\boldsymbol{r}_t^n = \sigma\left(\boldsymbol{W}_{\mathrm{in}}^{n,r} \boldsymbol{y}_t^{n-1} + \boldsymbol{W}_{\mathrm{rec}}^n \boldsymbol{r}_{t-1}^n\right) \tag{3}$$

where $\boldsymbol{r}^n$ denotes the hidden state, $\boldsymbol{W}_{\mathrm{in}}^{n,r}$ is the input weight matrix, $\boldsymbol{W}_{\mathrm{rec}}^n$ is the recurrent weight matrix, and $\sigma(\cdot)$ is the *sigmoid*$(\cdot)$ function. The LSTM module consists of four gates and one continuous variable, called cell state, which can be formulated as

$$\begin{cases} \boldsymbol{i}_t^{n,r} = \sigma\left(\boldsymbol{W}_{\mathrm{in,i}}^{n,r} \boldsymbol{y}_t^{n-1} + \boldsymbol{W}_{\mathrm{rec,i}}^n \boldsymbol{r}_{t-1}^n\right) \\ \boldsymbol{f}_t^n = \sigma\left(\boldsymbol{W}_{\mathrm{in,f}}^{n,r} \boldsymbol{y}_t^{n-1} + \boldsymbol{W}_{\mathrm{rec,f}}^n \boldsymbol{r}_{t-1}^n\right) \\ \boldsymbol{o}_t^n = \sigma\left(\boldsymbol{W}_{\mathrm{in,o}}^{n,r} \boldsymbol{y}_t^{n-1} + \boldsymbol{W}_{\mathrm{rec,o}}^n \boldsymbol{r}_{t-1}^n\right) \\ \boldsymbol{g}_t^n = \phi\left(\boldsymbol{W}_{\mathrm{in,g}}^{n,r} \boldsymbol{y}_t^{n-1} + \boldsymbol{W}_{\mathrm{rec,g}}^n \boldsymbol{r}_{t-1}^n\right) \\ \boldsymbol{c}_t^n = \boldsymbol{c}_{t-1}^n \odot \boldsymbol{f}_t^n + \boldsymbol{i}_t^n \odot \boldsymbol{g}_t^n \\ \boldsymbol{r}_t^n = \boldsymbol{o}_t^n \odot \phi\left(\boldsymbol{c}_t^n\right) \end{cases} \tag{4}$$

where $\boldsymbol{i}$,$\boldsymbol{f}$, and $\boldsymbol{o}$ denote the states of the input gate, the forgetting gate, and the output gate, respectively. $\boldsymbol{g}$, $\boldsymbol{c}$, $\boldsymbol{r}$ denote the candidate state, the cell state, and the hidden state, respectively. $\phi(\cdot)$ denotes the $\tanh(\cdot)$ function and $\odot$ is the Hadamard product. The LIF neuron simultaneously receives signals from the previous layer and the current layer for updating its membrane potential $\boldsymbol{u}$. When a neuron's membrane potential u_i exceeds a firing threshold u_{th}, the neuron fires a spike s_i and resets its membrane potential to u_0. The behaviours of the LIF module can be written as

$$\begin{cases} \tau \frac{d\boldsymbol{u}^n(t)}{dt} = -\boldsymbol{u}^n(t) + \boldsymbol{W}_{\mathrm{in}}^{n,s} \boldsymbol{y}^{n-1}(t) \\ \begin{cases} s_i^n(t) = 1 \,\&\, u_i^n(t) = u_0, & \text{if } u_i^n(t) \geq u_{th} \\ s_i^n(t) = 0, & \text{otherwise} \end{cases} \end{cases} \tag{5}$$

where $\boldsymbol{W}_{\mathrm{in}}^{n,s}$ is the input weight matrix. To make the continuous neural dynamics friendlier for programming the gradient-descent learning approaches, we further convert Eq. (5) into an explicitly iterative version[8] as

$$\begin{cases} \boldsymbol{u}_t^n = e^{-\frac{dt}{\tau}} \boldsymbol{u}_{t-1}^n \odot \left(1 - \boldsymbol{s}_{t-1}^n\right) + \boldsymbol{W}_{\mathrm{in}}^{n,s} \boldsymbol{y}_t^{n-1} \\ \boldsymbol{s}_t^n = \mathrm{H}\left(\boldsymbol{u}_t^n - u_{th}\right), \end{cases} \tag{6}$$

where $\mathrm{H}(\cdot)$ is the Heaviside function that satisfies $\mathrm{H}(x) = 1$ when $x \geq 0$ and $\mathrm{H}(x) = 0$ otherwise. Here we assume $u_0 = 0$.

Unlike the fixed firing threshold of the LIF neuron, the ALIF neuron further introduces adaptive thresholds. The evolution of the firing thresholds of the ALIF neuron, η, can be described as

$$\begin{cases} \boldsymbol{b}_t^n = \rho^n \boldsymbol{b}_{t-1}^n + (1 - \rho^n) \boldsymbol{s}_t^n \\ \boldsymbol{\eta}_t^n = 0.1 + \alpha^n \boldsymbol{b}_t^n, \end{cases} \tag{7}$$

where ρ^n denotes the learnable parameters that control the update rate of the adaptive thresholds. The parameter α^n is a constant that controls the size of adaptation of the thresholds, set to 0.2 by default.

Details of the three-stage learning for HSTNN

We develop a three-stage learning methodology to create the HSTNN progressively, including *Adaptation*, *Selection*, and *Restoration* stages.

***Adaptation* stage.** To learn the optimal hybrid connections, the adaptation stage expands each hybrid layer by two redundant neuron populations. In particular, to generate a hybrid layer with M neurons, it first introduces an SNN pool with M neurons and an RNN pool with M neurons. Each pool works in its respective dynamics and different types of signals are mixed by Eq. 1 before sending to the next layer. The synaptic weights are trained by the BPTT learning algorithm, producing a better starting point for the following *Selection* stage. In this way, the adaptation stage provides greater flexibility in exploiting the hybrid structure and integrating the distinct dynamic behaviours of RNNs and SNNs into a unified optimization framework.

***Selection* stage.** To select the optimal structure from the abundant pools, the *Selection* stage identifies and ranks the importance of neurons.

The neuronal importance is evaluated by aggregating the importance scores of its afferent weights. The weight score is evaluated based on a classical parameter saliency measure[16,38], which accesses the saliency of a parameter through calculating the smallest change of the loss function ΔL caused by perturbing the specific parameter.

Next, we will formulate the smallest change of the loss function caused by the perturbation as an optimization problem[16] and employ a neuron-wise pruning strategy to adapt the saliency measure for the hybrid model. A key relationship exists between the parameter perturbation and neuron pruning: pruning an unimportant neuron can be formalized as perturbing the model such that all weights connecting to the unimportant neuron become zero (i.e., $\Delta w = -w$, where Δw denotes the weight perturbation). On this basis, the change of ΔL, expressed in the Taylor expansion form, is governed by

$$L = \nabla_w L(w)^T w + \frac{1}{2} w^T H w + O(\| w\|^3). \tag{8}$$

Based on the OBS[16], we assume that a trained neural network model (i.e., the HSTNN established by the *Adaptation* stage) has converged to a local minimum of the loss function L, where the gradient yields $\nabla_w L(w) = 0$, and the Hessian matrix H is positive semidefinite. Thus, ΔL can be primarily associated with the second-order term containing the Hessian matrix $\Delta w^T H \Delta w$.

We then formulate the process of finding the smallest change of the loss function ΔL while removing the specific weight parameter w_u as an optimization problem:

$$\min_{\Delta w} \frac{1}{2}\Delta w^T H \Delta w = \frac{1}{2}\begin{pmatrix}\Delta w_u \\ \Delta w_i\end{pmatrix}^T \begin{pmatrix} H_{u,u} & H_{u,i} \\ H_{i,u} & H_{i,i}\end{pmatrix}\begin{pmatrix}\Delta w_u \\ \Delta w_i\end{pmatrix} \quad (9)$$
$$\text{s.t. } \Delta w_u + w_u = 0,$$

where w_u and w_i denote the weight groups of unimportant and important neurons, respectively. The Hessian matrix H can be further written as a block matrix. Given the importance of w_u is measured by how its removal influences the smallest change in the loss function, we set $\Delta w_u = -w_u$. Solving the above optimization problem using the Lagrangian method yields:

$$H_{i,i} w_i + H_{i,u} w_u = 0. \quad (10)$$

Substituting Δw_i by $w_i = H_{i,i}^{-1} H_{i,u} w_u$ results in a solution[16] to Eq. (9):

$$\frac{1}{2} w^T H w = \frac{1}{2} w_u^T (H_{u,u} - H_{u,i} H_{i,i}^{-1} H_{i,u}) w_u. \quad (11)$$

By employing the Schur complement of the inverse matrix, we have

$$\frac{1}{2} w_u^T (H_{u,u} - H_{u,i} H_{i,i}^{-1} H_{i,u}) w_u = \frac{1}{2} w_u^T \left[H^{-1}\right]_{u,u}^{-1} w_u. \quad (12)$$

The original OBS requires measuring the perturbation estimation for all parameters separately and calculating the matrix inverse of the Hessian matrix, which leads to an intolerable computational cost in large-scale neural networks. Instead, we focus on evaluating the comprehensive impact of a group of afferent weights connecting to the same neuron. Therefore, we develop a neuron-wise strategy based on the structural pruning method[17]. Specifically, we first group all weight parameters connecting to a specific output neuron and compute the corresponding perturbation when this group of weight is pruned. Additionally, following previous studies[17,38,39], we assume the main saliency features are contained within the diagonal blocks. Therefore, the Hessian matrix H can be approximated by a diagonal block matrix where each block includes a diagonal operator:

$$\frac{1}{2} w_u^T \left[H^{-1}\right]_{u,u}^{-1} w_u \approx \frac{1}{2} w_u^T H_{u,u} w_u \approx \frac{1}{2} w_u^T \frac{\mathrm{Tr}(H_{u,u})}{u} w_u = \frac{\mathrm{Tr}(H_{u,u})}{2u} \parallel w_u \|_2^2 \quad (13)$$

where $\mathrm{Tr}(H_{u,u})$ denotes the trace of the block diagonal Hessian of the unimportant group. The above equation effectively avoids the computation of the inverse of the Hessian matrix by using the trace of $\mathrm{Tr}(H_{u,u})$. Furthermore, we employ the Hutchinson method[40,41] for calculating the trace, which employs stochastic vectors to effectively estimate the Hessian operator (see Eqs 6, 7 in reference[41] for implementations).

Consequently, the neuronal importance score, measured by the above smallest perturbation, can be evaluated by the above Hessian trace estimation with only a moderate computational cost. Considering the distinct neuronal dynamics and representation manners between RNNs and SNNs, we rank the importance scores of spiking and non-spiking neurons, separately. Specifically, we collect the same types of neurons from all layers and uniformly sort them according to the important scores. Given the ranking results, we select a certain percentage of neurons from each pool as important neurons according to the predefined SNN ratio.

***Restoration* stage.** Given the ranking results, the *Restoration* stage further prunes redundant neurons and their inactive connections and fine-tunes the resulting compact network. To this end, we create the corresponding binary mask matrix for the specified input weight connections, $\boldsymbol{W}_{\text{in}}^{n,r}$ and $\boldsymbol{W}_{\text{in}}^{n,s}$, and the recurrent weight connections of RNNs, $\boldsymbol{W}_{\text{rec}}^{n}$, based on the indices of selected neurons.

Formally, let the total number of neurons in layer n be l_n, the index set of selected artificial neurons be $R(n) := \{i_1^n, \ldots, i_k^n\}$ and the index set of the selected spiking neurons be $S(n) := \{j_1^n, \ldots, j_{k'}^n\}$. Let $\boldsymbol{1}_i$ be a column unit vector with the i-th element being 1. The size of $R(n)$ and $S(n)$ are denoted as $|R(n)|$ and $|S(n)|$, respectively. The mask matrix $\boldsymbol{m}^{n,r} \in \mathbb{R}^{l_{n-1} \times |R(n)|}$ for non-spiking neurons can be formalized using unit vectors and ordering the indices in $R(n)$ following an ascending order:

$$\boldsymbol{m}^{n,r} = [\boldsymbol{1}_{i_1^n}, \ldots, \boldsymbol{1}_{i_k^n}], \quad i_1^n, \ldots, i_k^n \in R(n). \quad (14)$$

Similarly, the mask matrix $\boldsymbol{m}^{n,s} \in \mathbb{R}^{l_{n-1} \times |S(n)|}$ for spiking neurons can be formalized in a similar way:

$$\boldsymbol{m}^{n,s} = [\boldsymbol{1}_{j_1^n}, \ldots, \boldsymbol{1}_{j_{k'}^n}], \quad j_1^n, \ldots, j_{k'}^n \in S(n). \quad (15)$$

We can then derive the concatenated mask matrix $\boldsymbol{m}^{n-1} \in \mathbb{R}^{l_{n-1} \times (|S(n)| + |R(n)|)}$ for both $\boldsymbol{W}_{\text{in}}^{n,r}$ and $\boldsymbol{W}_{in}^{n,s}$

$$\boldsymbol{m}^{n-1} = \begin{pmatrix} \boldsymbol{m}^{n-1,s} & 0 \\ 0 & \boldsymbol{m}^{n-1,r}\end{pmatrix}. \quad (16)$$

Given $\boldsymbol{m}^{n,r}$, $\boldsymbol{m}^{n,s}$, and $\boldsymbol{m}^{n-1}$, we derive the shrinked weights, $\boldsymbol{W}_{\text{in}}^{\prime n,r}$, $\boldsymbol{W}_{\text{in}}^{\prime n,s}$, $\boldsymbol{W}_{\text{rec}}^{\prime n}$ in the n-th layer after the *Restoration* stage by

$$\begin{cases} \boldsymbol{W}_{\text{in}}^{\prime n,r} \leftarrow (\boldsymbol{m}^{n,r})^T \boldsymbol{W}_{\text{in}}^{n,r} \boldsymbol{m}^{n-1} \\ \boldsymbol{W}_{\text{in}}^{\prime n,s} \leftarrow (\boldsymbol{m}^{n,s})^T \boldsymbol{W}_{\text{in}}^{n,s} \boldsymbol{m}^{n-1} \\ \boldsymbol{W}_{\text{rec}}^{\prime n} \leftarrow (\boldsymbol{m}^{n,r})^T \boldsymbol{W}_{\text{rec}}^{n} \boldsymbol{m}^{n,r} \end{cases}. \quad (17)$$

After that, we retrain the final compact HSTNN to fine-tune the parameters.

Details of the learning algorithm for HSTNN. BPTT[4] is a powerful learning algorithm for RNNs and recently has been adapted to train SNNs by addressing the convergence problem and the non-differentiable spiking activities[8,12,29]. The training approaches for both RNNs and SNNs share several core features, including the backpropagation of gradients through spatial (layer-wise) and temporal (time step-wise) dimensions, and the subsequent update of parameters based on these gradients across all time steps. Given these similarities, we employ a unified BPTT methodology, incorporating the surrogate function for spiking activities, to train the HSTNN. We introduce the notation δ for the gradient regarding the loss function L, for example, $\delta o = \frac{\partial L}{\partial o}$. For a vanilla RNN module, we have

$$\delta \boldsymbol{r}_t^n = \left(\boldsymbol{W}_{\text{in}}^{n+1,r}\right)^T (\sigma' \odot \delta \boldsymbol{r}_t^{n+1}) + \left(\boldsymbol{W}_{\text{rec}}^{n}\right)^T (\sigma' \odot \delta \boldsymbol{r}_{t+1}^n) \quad (18)$$

where σ' represents the gradient of the activation function. For the LIF-based SNN module, we have

$$\begin{cases} \delta \boldsymbol{s}_t^n = \left(\boldsymbol{W}_{\text{in}}^{n+1,s}\right)^T \delta \boldsymbol{u}_t^{n+1} - e^{-\frac{dt}{\tau}} \boldsymbol{u}_t^n \odot \delta \boldsymbol{u}_{t+1}^n \\ \delta \boldsymbol{u}_t^n = \mathrm{H}' \odot \delta \boldsymbol{s}_t^n + e^{-\frac{dt}{\tau}} (1 - \boldsymbol{s}_t^n) \odot \delta \boldsymbol{u}_{t+1}^n \end{cases} \quad (19)$$

where H' is the gradient of the Heaviside function, which is actually non-differentiable. To solve this problem, we use the surrogate function to approximate its gradient[29]. An empirical analysis of the effect of specific surrogate function formats on the Hessian trace, used in the *Selection* stage, is provided in Supplementary Fig. 2. The gradient expressions for more complex neuronal modules are similar to those in Eqs. 18, 19 and thus are omitted here for clarity.

Details of parameter configurations and model comprehensive evaluation

We used consistent network structures for SNNs, RNNs, directly-hybrid models, and HSTNNs in Fig. 2. On N-MNIST, S-MNIST, and PTB datasets, the network structures of [input-800-800-10], [input-800-800-10], and [input-650-650-10,000] were employed, respectively, to compare the task performance of different models. On the above three datasets, the HSTNNs were built based on vanilla RNN and LIF models. Please note that in our experiments, the network structure consistently refers to the network structure after the three-stage learning. By default, an equal number of spiking and non-spiking neurons are utilized in the selection stage unless stated otherwise. On DVS-Gesture, the network structure of [input-128C3-AP2-256C3-AP2-384C3-AP2-256-11] was adopted, using recurrent convolutional neural network (RCNN) and LIF-based spiking convolutional neural network models for HSTNN construction. The implementation of RCNN and RSNN followed the formulations in Eqs. 3–6 but with the simple operation of weighted sum replaced by the convolutional operation. For RCNN and RSNN, we adapted the selection process by using a structural grouping strategy that selects the most important output feature map channels based on cumulative importance scores across all neurons within the same feature map. The selected feature maps are then maintained to create a reduced network structure for retraining in the *Restoration* stage. The SGD optimizer was chosen for the PTB dataset, while Adam was for the S-MNIST, N-MNIST, and DVS-gesture datasets. Detailed parameter configurations are provided in Supplementary Table 1.

We employed the consistent loss functions across three learning stages on all datasets. For the language modeling task, we utilized a cross-entropy-based loss function, which can be formalized by

$$\mathrm{L} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{l_N}\boldsymbol{g}_{t,i}\log\left(\hat{\boldsymbol{y}}_{t,i}^{N}\right) \tag{20}$$

where $\boldsymbol{g}_t$ is a one-hot vector that denotes the real distribution of vocabularies and $\hat{\boldsymbol{y}}_t^N = \mathrm{softmax}(\boldsymbol{y}_t^N)$ denotes the predicted distribution at the t-th time step. The most recent spiking temporal information was used in Eq. 2 for computing $\boldsymbol{y}_{t,i}^N$. For S-MNIST, a similar cross-entropy loss was used:

$$\mathrm{L} = -\sum_{i=1}^{l_N}\boldsymbol{y}_i\log\left(\boldsymbol{y}_{T,i}^{N}\right). \tag{21}$$

where the rate coding was used in Eq. 2 for computing $\boldsymbol{y}_{T,i}^N$.

For classification tasks on neuromorphic datasets including N-MNIST and DVS-Gesture, we used the Mean Squared Error loss function:

$$\mathrm{L} = \frac{1}{l_N}\sum_{i=1}^{l_N}\left(\boldsymbol{y}_i - \boldsymbol{y}_{T,i}^{N}\right)^2. \tag{22}$$

where l_N denotes the number of neurons in the layer N and the rate coding was used for computing $\boldsymbol{y}_{T,i}^N$.

The computational cost was evaluated at the operation level. For a vanilla RNN module with M_i input neurons and M_o output neurons, the computational cost can be estimated as:

$$\begin{aligned} C_{RNN} &\approx \left(M_iM_o\left(C_{mul}+C_{add}\right)+M_oM_o\left(C_{mul}+C_{add}\right)\right)T \\ &= \left(\left(M_iM_o+M_oM_o\right)\left(C_{mul}+C_{add}\right)\right)T \end{aligned} \tag{23}$$

where C_{mul} and C_{add} denote the basic computational costs of a multiplication operation and an addition operation, respectively, and T denotes the number of time steps. In order to provide an intuitive and concise comparison, here we mainly estimated the computational cost of matrix operations, which produces a great impact on the hardware execution energy, and ignored the computation of the vector or scalar operations. For our implementation of the LIF-based SNN module, there is no recurrent matrix computation, and the multiplication operations can be replaced with sparse accumulation operations benefiting from the binary spike format. We thereby evaluated the cost of a LIF-based SNN module by

$$C_{SNN} \approx sC_{add}M_iM_oT \tag{24}$$

where s denotes the average spike rate during the entire inference stage (normalized within [0, 1]). As with RNNs, the computational cost of the vector or scalar operations is omitted for clarity. Since more and more neuromorphic chips[11,28] efficiently support the hybrid execution between non-spiking computation and spiking computation, the computational cost of a hybrid layer can be derived based on the results of single-paradigm RNN or SNN modules. Assuming that there are M_{i1} non-spiking inputs, M_{i2} spiking inputs, M_{o1} RNN output neurons, and M_{o2} SNN output neurons, the computational cost of an HSTNN layer yields

$$C_{HSTNN} \approx [(M_{i1}(M_{o1}+M_{o2})+M_{o1}M_{o1})(C_{mul}+C_{add})+sM_{i2}(M_{o1}+M_{o2})C_{add}]T \tag{25}$$

where C_{HSTNN} is smaller than C_{RNN} owing to the insertion of the SNN with a much lower computational cost. The estimation for other more complicated neuron models is similar by incorporating more matrix operations and we omit them for clarity.

Details of experimental setup for the robustness evaluation

HSTNNs were constructed using the optimal SNN ratio reported in Fig. 2 for comparison: SNN ratios of 0.25, 0.95, and 0.75 for S-MNIST, N-MNIST, and DVS-Gesture datasets, respectively. All models were trained on standard training sets and evaluated on preprocessed testing sets. Three types of model robustness were evaluated: random noise robustness, frame-loss robustness, and adversarial attack robustness. On the S-MNIST datasets, the network structures [input-400-400-10] were employed. The same structures for N-MNIST and DVS-Gesture as those used in Fig. 2 were employed in the comparison.

In Fig. 3b, for the random noise robustness, we added the Gaussian noise with a zero mean and a 0.05 standard deviation into each testing sample of S-MNIST and added the salt-and-pepper noise into each testing sample of N-MNIST with a probability of 0.1. For the frame-loss robustness, we randomly masked some sequence information of each frame of the testing sample with a probability of 0.1. For the adversarial attack robustness, we generated the untargeted adversarial sample ($\boldsymbol{x}'$) by adding an imperceptible perturbation ($\boldsymbol{\delta}$) into the raw testing sample ($\boldsymbol{x}$)[42]. The perturbation can be defined by

$$\mathrm{argmin}_{\boldsymbol{\delta}}\|\ \boldsymbol{\delta}\ \|_2, \quad s.t., f(\boldsymbol{x}+\boldsymbol{\delta}) \neq f(\boldsymbol{x}), \tag{26}$$

where $f(\boldsymbol{x})$ generally refers to the output of the victim model. To solve the above optimization problem, we followed the prior work[42] and took an iterative strategy to calculate the gradient with respect to the spike input sample ($\boldsymbol{xs}$) as follows:

$$\delta\boldsymbol{s}_i = {}_{\boldsymbol{xs}_i}L(\theta, \boldsymbol{xs}_i, \boldsymbol{y}_{original}), \tag{27}$$

where $\delta\boldsymbol{s}_i$ represents the input gradient at the ith iteration. Since the elements in $\delta\boldsymbol{s}_i$ are continuous values, in order to generate the spike-based adversarial input $\boldsymbol{xs}_i'$, we used a two-stage method proposed by Liang et al.[14], called gradient-to-spike (G2S) and restricted spike flipper (RSF). Specifically, the G2S technique was used to convert the continuous gradient into a ternary one (i.e., $\{-1, 0, 1\}$) via probabilistic sampling from the normalization version of δs_i:

$$\begin{aligned}\delta s_i' &= \mathrm{sign}(\boldsymbol{\delta}_{mask} \odot \delta \boldsymbol{s}_i), \quad P(\boldsymbol{\delta}_{mask} = 1)\\ &= \mathrm{norm}(|\delta \boldsymbol{s}_i|), \quad P(\boldsymbol{\delta}_{mask} = 0) = 1 - \mathrm{norm}(|\delta \boldsymbol{s}_i|),\end{aligned} \tag{28}$$

where $\boldsymbol{\delta}_{mask}$ is a binary mask and norm(·) is a scaling normalization function that normalizes each element into the range of [0, 1]. Then an overflow-aware transformation was utilized to avoid the overflow of the resulting $\boldsymbol{xs}_i$, i.e., keeping the resulting $\boldsymbol{xs}_i'$ as a binary spike within {0, 1}. The entire G2S process can be described as

$$\boldsymbol{xs}_i' = \mathrm{transform}(\delta \boldsymbol{s}_i', \boldsymbol{xs}_i). \tag{29}$$

The RSF technique was used to address the gradient vanishing problem. When meeting all-zero input gradients, the spiking inputs can be flipped randomly with a control of the turnover rate. We ran 20 iterations to generate each adversarial sample.

Details of the experimental setup for the scalability evaluation

In Fig. 3d, we demonstrated the combinations of vRNN&LIF, vRNN&ALIF, and LSTM&LIF using the same network structure (i.e., [input-400-400-10]). For the RCNN&SCNN, we applied the network structure as utilized in Fig. 2d. An optimal SNN ratio of 0.75, which yielded the best classification accuracy, was selected for constructing the HSTNNs. In Table 1, a network structure of [input-128C3-AP2-256C3-AP2-384C3-256-10] was used for N-MNIST, and a structure of [input-128C3-128C3-AP2-128C3-128C3-AP2-256C3-256C3-AP2-512C3-512C3-512C3-512C3-10] was employed for CIFAR10-DVS. Optimal SNN ratios of 0.875 for both N-MNIST and CIFAR10-DVS were adopted for the construction of HSTNNs. We provided other parameter settings and training details in Supplementary Table 1.

Details of experiments on the robot place recognition

We conducted robot navigation in three different environments: an indoor environment with adequate lighting (env1), an outdoor environment with varying lighting conditions (env2), and an indoor environment with low lighting (env3). The robot traversed a predefined path six times in each environment, collecting event-based data using a DVS camera and frame-based data using an RGB camera. The path was divided into 100 segments representing distinct places, and the objective was to recognize the current scenario within these 100 classes. For data preprocessing, we utilized a pre-trained four-layer CNN and a four-layer SCNN, as described in prior work[30], to handle the inputs from the RGB and DVS cameras, respectively, used for robot place recognition. The CNN used for RGB images, processed inputs of size 240 × 180 × 3 to include three RGB channels. The SCNN processed event images with an input size of 240 × 180 pixels, incorporating both positive and negative polarity information. The parameters of both pre-trained CNN and SCNN were fixed in our simulations. Outputs from the CNN and SCNN models were combined and fed into a three-layer HSTNN with a network structure of [input-500-500-100]. Due to the different resolutions of DVS and RGB cameras, we used nine consecutive event images and three corresponding RGB images as a training sample. The HSTNN was then constructed through our three-stage hybrid approach, learning to recognize the correct place from among 100 candidates. The training process involved 150 epochs for the *Selection* stage and 100 epochs for the *Restoration* stage. We employed the Adam optimizer and a cross-entropy loss function for the three-stage learning stage. To evaluate computational cost, we analyzed the overall number of operations performed by the hybrid modules by utilizing Eqs. 23–25.

Details of implementation on neuromorphic hardware

TianjicX is a hybrid neuromorphic chip that can flexibly allocate computing resources and schedule execution time for multiple neural network tasks, including both ANNs and SNNs[34]. However, the flexibility of the chip also complicates the deployment of neural networks. We describe the mapping details when deploying HSTNNs on TianjicX from a top-down perspective as follows.

At the network level, layers were first grouped and mapped to core groups, where the number of cores depends on the structure and the computational cost of the layers. In the experiment, we assigned a core group for each layer in HSTNNs. Core groups can run in a pipeline manner on the TianjicX chip as depicted in Supplementary Fig. 3b. The reported results were collected in the scenario of running a single sample. The layer-level mapping strategies are illustrated in Fig. 5b. We applied different mapping strategies for different layers considering the layer size. In the fixed-core mapping, a layer was mapped onto a core group containing a small fixed number of cores dedicated to computing the RNN or SNN module, respectively. The workload for each core varies according to the SNN ratio. We used this strategy for small layers, such as those on S-MNIST, and fixed both the core numbers for SNN or RNN modules to one. This is because partitioning a small layer cannot fully utilize the parallelism of multiple cores but brings additional data transfer, which can result in resource wastage and excessive power consumption. In contrast, larger layers can utilize the resources of multiple cores better, so we used more cores with a fixed workload for each and allocated them for the SNN or RNN module according to the SNN ratio. The TianjicX chip supports a primitive instruction set that covers a wide range of operations. To perform computation of the layers, we configured the primitive sequence for each core. The operations required by HSTNNs were listed in Supplementary Fig. 3c. We used 8-bit integers for the outputs of both RNN and SNN populations, thus simplifying the output concatenation in each layer.

Following the mapping steps above, we successfully implemented HSTNNs on the TianjicX development board (see Supplementary Fig. 3d). The execution latency and energy consumption results shown in Supplementary Fig. 3a validated the efficiency and flexibility of HSTNNs. It is worth mentioning that the execution latency of HSTNNs can be shorter than that of the sole SNN. For the networks on S-MNIST, we noticed that although a sole SNN (with the SNN ratio of 1) has the least computational workload, it doesn't achieve minimal latency due to the utilization of only a single core. Conversely, for hybrid models, as the SNN ratio decreases, the latency of the SNN core shortens, and that of the RNN core lengthens. The total latency is the maximum of the latencies consumed by the two cores, thus achieving the minimum value when their latencies are equal. In the case of the networks on N-MNIST, we found that the latencies at SNN ratios of 0.75 and 1 were almost identical, possibly because the minimal additional latency of the small RNN module in the first layer at the SNN ratio of 0.75 may be offset by the data transfer latency and the off-chip measurement we adopted might introduce errors.

Data availability

All data used in this paper are publicly available. The S-MNIST and MNIST datasets are available at http://yann.lecun.com/exdb/mnist/. The PTB dataset is available at https://catalog.ldc.upenn.edu/docs/LDC95T7/. The DVS-Gesture dataset is available at https://ibm.ent.box.com/s/3hiq58ww1pbbjrinh367ykfdf60xsfm8. The CIFAR10-DVS dataset is available at https://figshare.com/articles/dataset/CIFAR10-DVS_New/4724671/2. The N-MNIST dataset is available at https://www.garrickorchard.com/datasets/n-mnist. The NeuroGPR dataset, used for place recognition tasks, can be accessed at https://zenodo.org/record/7845007/. The Multi Vehicle Stereo Event Camera (MVSEC) dataset is available at https://daniilidis-group.github.io/mvsec/. Source data are provided with this paper.

Code availability

Source codes for reproducing the results in this paper are available at https://github.com/shibizhao/hstnn-demo, with Zenodo link https://zenodo.org/records/13166818[43].

References

1. Graves, A., Mohamed, A.-R. & Hinton, G. Speech recognition with deep recurrent neural networks. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 6645–6649 (IEEE, 2013).
2. Sun, T.-X., Liu, X.-Y., Qiu, X.-P. & Huang, X.-J. Paradigm shift in natural language processing. *Mach. Intell. Res.* **19**, 169–183 (2022).
3. Graves, A. et al. Hybrid computing using a neural network with dynamic external memory. *Nature* **538**, 471–476 (2016).
4. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986).
5. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Comput.* **9**, 1735–1780 (1997).
6. Maass, W. Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* **10**, 1659–1671 (1997).
7. Gerstner, W., Kistler, W. M., Naud, R. & Paninski, L. *Neuronal dynamics: From Single Neurons to Networks and Models of Cognition* (Cambridge University Press, 2014).
8. Wu, Y. et al. Direct training for spiking neural networks: Faster, larger, better. In *Proc. AAAI Conference on Artificial Intelligence*, **33**, 1311–1318 (AAAI, 2019).
9. Wu, J., Yılmaz, E., Zhang, M., Li, H. & Tan, K. C. Deep spiking neural networks for large vocabulary automatic speech recognition. *Front. Neurosci.* **14**, 199 (2020).
10. Chu, H. et al. A neuromorphic processing system for low-power wearable ECG classification. In *2021 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 1–5 (IEEE, 2021).
11. Tian, L., Wu, Z., Wu, S. & Shi, L. Hybrid neural state machine for neural network. *Sci. China Inf. Sci.* **64**, 1–13 (2021).
12. He, W. et al. Comparing SNNs and RNNs on neuromorphic vision datasets: similarities and differences. *Neural Netw.* **132**, 108–120 (2020).
13. Lichtsteiner, P., Posch, C. & Delbruck, T. A 128 × 128 120 db 15 us latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits* **43**, 566–576 (2008).
14. Liang, L. et al. Exploring adversarial attack in spiking neural networks with spike-compatible gradient. *IEEE Trans. Neural Netw. Learn. Syst.* **34**, 2569–2583 (2021).
15. Deng, L. et al. Rethinking the performance comparison between SNNs and RNNs. *Neural Netw.* **121**, 294–307 (2020).
16. Hassibi, B., Stork, D. G. & Wolff, G. J. Optimal brain surgeon and general network pruning. In *Proc. IEEE International Conference On Neural Networks*, 293–299 (IEEE, 1993).
17. Yu, S. et al. Hessian-aware pruning and optimal neural implant. In *Proc. IEEE/CVF Winter Conference on Applications of Computer Vision*, 3880–3891 (IEEE, 2022).
18. Rathi, N., Srinivasan, G., Panda, P. & Roy, K. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *Proc. International Conference on Learning Representations* (ICLR, 2019).
19. Datta, G., Kundu, S. & Beerel, P. A. Training energy-efficient deep spiking neural networks with single-spike hybrid input encoding. In *Proc. International Joint Conference on Neural Networks (IJCNN)*, 1–8 (IEEE, 2021).
20. Ponghiran, W. & Roy, K. Hybrid analog-spiking long short-term memory for energy-efficient computing on edge devices. In *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 581–586 (IEEE, 2021).
21. Yang, Q. et al. Training spiking neural networks with local tandem learning. *Adv. Neural Inf. Process. Syst.* **35**, 12662–12676 (2022).
22. Xu, Q. et al. Constructing deep spiking neural networks from artificial neural networks with knowledge distillation. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7886–7895 (IEEE, 2023).
23. Zhao, R. et al. A framework for the general design and computation of hybrid neural networks. *Nat. Commun.* **13**, 1–12 (2022).
24. Negi, S., Sharma, D., Kosta, A. K. & Roy, K. Best of both worlds: Hybrid SNN-ANN architecture for event-based optical flow estimation. *arXiv e-prints* arXiv–2306 (2023).
25. Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A. & Ganjtabesh, M. First-spike-based visual categorization using reward-modulated stdp. *IEEE Trans. Neural Netw. Learn. Syst.* **29**, 6178–6190 (2018).
26. Diehl, P. U. & Cook, M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* **9**, 99 (2015).
27. Davies, M. et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* **38**, 82–99 (2018).
28. Orchard, G. et al. Efficient neuromorphic signal processing with loihi 2. In *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, 254–259 (IEEE, 2021).
29. Wu, Y., Deng, L., Li, G., Zhu, J. & Shi, L. Spatio-temporal back-propagation for training high-performance spiking neural networks. *Front. Neurosci.* **12**, 331 (2018).
30. Yu, F. et al. Brain-inspired multimodal hybrid neural network for robot place recognition. *Sci. Robot.* **8**, eabm6996 (2023).
31. Pei, J. et al. Towards artificial general intelligence with hybrid Tianjin chip architecture. *Nature* **572**, 106–111 (2019).
32. Höppner, S. et al. The Spinnaker 2 processing element architecture for hybrid digital neuromorphic computing. *arXiv preprint arXiv:2103.08392* (2021).
33. Pehle, C. et al. The brain scales-2 accelerated neuromorphic system with hybrid plasticity. *Front. Neurosci.* **16** (2022).
34. Ma, S. et al. Neuromorphic computing chip with spatiotemporal elasticity for multi-intelligent-tasking robots. *Sci. Robot.* **7**, eabk2948 (2022).
35. Hassabis, D., Kumaran, D., Summerfield, C. & Botvinick, M. Neuroscience-inspired artificial intelligence. *Neuron* **95**, 245–258 (2017).
36. Panzeri, S., Brunel, N., Logothetis, N. K. & Kayser, C. Sensory neural codes using multiplexed temporal scales. *Trends Neurosci.* **33**, 111–120 (2010).
37. Bellec, G., Salaj, D., Subramoney, A., Legenstein, R. & Maass, W. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in neural information processing systems.* **31** (2018).
38. LeCun, Y., Denker, J. & Solla, S. Optimal brain damage. *Advances in neural information processing systems***2** (1989).
39. Liu, C., Zhang, Z. & Wang, D. Pruning deep neural networks by optimal brain damage. *Interspeech*, 1092–1095 (2014).
40. Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Commun. Stat.-Simul. Comput.* **18**, 1059–1076 (1989).
41. Dong, Z. et al. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Adv. neural Inf. Process. Syst.* **33**, 18518–18529 (2020).
42. Kurakin, A., Goodfellow, I. & Bengio, S. Adversarial examples in the physical world. In *Artificial Intelligence Safety and Security* 99–112 (Chapman and Hall/CRC, 2018).
43. Shi, B. et al. Adaptive spatiotemporal neural networks through complementary hybridization. https://doi.org/10.5281/zenodo.13166818.

44. Pei, Y., Xu, C., Wu, Z., Liu, Y. & Yang, Y. Albsnn: ultra-low latency adaptive local binary spiking neural network with accuracy loss estimator. *Front. Neurosci.***17,** 1225871 (2023).
45. 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).
46. Fang, W. et al. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proc. IEEE/CVF International Conference on Computer Vision*, 2661–2671 (*IEEE,* 2021).
47. Wu, Y. et al. Brain-inspired global-local learning incorporated with neuromorphic computing. *Nat. Commun.* **13**, 65 (2022).
48. Zheng, H., Wu, Y., Deng, L., Hu, Y. & Li, G. Going deeper with directly-trained larger spiking neural networks. In *Proc. AAAI Conference on Artificial Intelligence*, vol. 35, 11062–11070 (*AAAI,* 2021).

Acknowledgements

We are grateful to Shixing Yu from Cornell University for his valuable comments on this research. This work was partially supported by the National Natural Science Foundation of China (No. 62106119, 62276151, 62090021), STI 2030 – Major Projects 2021ZD0200300, CETC Haikang Group-Brain Inspired Computing Joint Research Center, the Hong Kong Polytechnic University under Project P0050631, and Chinese Institute for Brain Research, Beijing.

Author contributions

L.D. and Y.W. conceived the work. Y.W., B.S., and F.Y. carried out the simulation experiments. Z.Z. and X.L. carried out the hardware implementation. Y.W., B.S., Z.Z., H.Z, G.L., and L.D. contributed to the analyses of experimental results. All of the authors contributed to the discussion of model and experiment design, and L.D. led the discussion. Y.W., B.S., Z.Z., and L.D. contributed to the writing of the paper. L.D. supervised the whole project.

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/s41467-024-51641-x.

Correspondence and requests for materials should be addressed to Lei Deng.

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