nature communications

Article

https://doi.org/10.1038/s41467-025-65197-x

Neuromorphic computing paradigms enhance robustness through spiking neural networks

Received: 1 May 2024

Accepted: 8 October 2025

Published online: 19 November 2025

Check for updates

Jianhao Ding[1], Zhaofei Yu[2], Jian K. Liu[3] & Tiejun Huang[1]

The success of deep learning methods over the past decade has been partially shrouded in the shadow of adversarial attacks. Even a tiny undetectable deformation can lead to vicious misleading targeted at safety-critical applications. In contrast, the brain is far more robust when performing complex cognitive tasks. Nevertheless, the underlying mechanisms that contribute to the brain's high reliability remain largely unexplored. At the intersection of neuroscience and artificial intelligence, we show that neuromorphic paradigms offer a promising solution to the dilemma brought by deep learning's inherent vulnerabilities. Specifically, we exploit the temporal processing capabilities of spiking neural networks (SNNs) to achieve robustness surpassing that of traditional artificial neural networks (ANNs). We demonstrate that prioritizing task-critical information in the encoded sequence and employing early exit decoding to ignore later perturbations significantly enhance SNN robustness. Further improvements in robustness are achieved by accurately capturing temporal dependencies through specialized training algorithms. Additionally, we introduce a fusion encoding strategy to balance SNN generalization on natural data with robustness against adversarial input. Experimental results on the CIFAR-10 dataset show that SNNs trained with these combined methods achieve twice the robustness of ANNs. Overall, our work demonstrates that neuromorphic computing, leveraging the temporal processing capabilities of SNNs, not only provides superior robustness compared to ANNs but also retains the benefit of low energy consumption. These advancements pave the way for developing next-generation, environmentally friendly, and reliable spike-based intelligent systems.

Our brains possess the remarkable ability to robustly perform a diverse range of functions, even in the face of ever-changing environments[1]. In contrast, current deep learning models used in Artificial Neural Networks (ANNs) lack this robustness, particularly under adversarial attacks. Even minor modifications to input images that are readily apparent to the human eye can cause ANNs to produce inaccurate predictions[2]. This vulnerability poses serious risks in safety-critical applications such as autonomous driving and human-robot interaction. Thus, adversarial attacks are seen as one of the most significant barriers to deploying deep models. In contrast, the brain's ability to perform complex cognitive tasks with remarkable reliability remains a puzzle.

One prominent distinction between ANNs and the brain lies in their temporal processing capabilities. While ANNs employ rate-based

[1]State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University, Beijing, China. [2]Institute for Artificial Intelligence, Peking University, Beijing, China. [3]School of Computer Science, University of Birmingham, Birmingham, UK. e-mail: yuzf12@pku.edu.cn

coding to simulate neuronal systems[3], the brain is far more complex, where precise spike timing plays a pivotal role[4–10]. Neuromorphic paradigms, built on large and programmable integrated circuits[11,12], enable spiking neural networks (SNNs) to simulate brain-like processing mechanisms, offering innovative solutions to challenges that ANNs struggle to address. SNNs provide two major advantages in temporal processing. First, their ability to encode information over durations allows for greater complexity in information representation, enabling richer and more nuanced temporal dynamics. This contrasts with the static nature of ANNs, which lacks the flexibility to handle temporal dependencies effectively. Second, SNNs utilize training algorithms designed to exploit precise spike timing, which often support global gradient backpropagation. This enables SNNs to process temporal information with high precision and adaptability[13–15]. These attributes make SNNs particularly suited for tasks handling fine-grained dynamic inputs. Furthermore, SNN-specific training algorithms (Fig. 1A) have enabled deep SNNs to tackle tasks traditionally reserved for deep ANNs, achieving comparable performance with significantly lower energy consumption. This emphasizes the growing potential of SNNs as an energy-efficient alternative for complex computational tasks.

Our work focuses on leveraging the unique capabilities of SNNs to address robustness challenges that have been difficult for ANNs to overcome. While previous research has demonstrated that SNNs can achieve robustness beyond that of ANNs, many approaches fail to fully exploit SNNs' temporal processing capabilities. In our experiments, we observed that adjusting the timing of spike emissions significantly influences SNN robustness. Specifically, we found that rate-coded SNNs can disregard perturbation on task-irrelevant background information during early network simulation, thereby enhancing robustness.

Building on these observations, we proposed a novel encoding method for SNNs that prioritizes task-critical information within the encoded sequence (Fig. 1B). This approach improves SNN robustness by ensuring that essential information is processed early. Furthermore, we discovered that SNN training algorithms capable of accurately capturing timing dependencies can guide SNNs to converge to network parameters that achieve robust generalization. These algorithms also support early exiting decoding, enhancing robustness by ignoring temporal perturbations occurring after the decoding exit. To address more complex perturbation types, we introduced a fusion encoding strategy that balances generalization on natural data with robustness to adversarial data. Our experiments on the CIFAR-10 dataset demonstrated that SNNs trained with fused encoding achieve approximately twice the accuracy of ReLU-based ANNs with the same architecture on attacked datasets.

Overall, our research demonstrates that SNNs can substantially enhance robustness by fully leveraging their temporal processing capabilities. This advancement not only positions SNNs as a more robust alternative to ANNs but also broadens their applicability in energy-efficient and safety-critical domains (Fig. 1C).

Results

Prioritizing task-critical temporal information during encoding boosts SNN robustness

SNNs have demonstrated a certain degree of robustness against adversarial attacks compared to ANNs when utilizing rate coding[16,17]. To examine if robustness stems from SNNs' temporal processing capabilities, we compare ANN and SNN models via ANN-SNN conversion, ensuring a fair comparison by sharing identical weight matrices. According to conversion theory, SNNs encode information through their spike count rate[18], which approximates the activation patterns of ANNs over a fixed duration[19]. In SNNs, predictions are derived by averaging the output spike count rate or postsynaptic potential (Fig. 2A), where the spike count rate refers to the total number of spikes emitted by a single neuron over a fixed duration. Unless stated otherwise, the term "rate" in this study denotes the spike count rate.

We train a three-layer ANN with 100 hidden neurons and ReLU activation to classify images from the MNIST dataset, and convert it into an SNN using integrate-and-fire (IF) neurons, following the ANN-SNN conversion process (see "Methods" section for details). Additionally, ANNs with the same architecture are trained as attackers to generate adversarial examples from the MNIST dataset using the fast gradient sign method (FGSM), with the attack intensity parameter ϵ set to 0.1 by default (see "Attack methods" section for details).

We first investigate the robustness of converted SNNs using two widely adopted rate encoding strategies: Poisson encoding[16,19] and current encoding[20,21]. The Poisson encoding maps pixel values within [0, 1] to spike trains sampled from Bernoulli distributions, while the current encoding inputs a continuous constant "current" directly into the SNN, resembling actual electrical currents and achieving higher accuracy. Fig. 2B illustrates differences in encoded sequences for clean and adversarial inputs. The current encoding repeats input perturbations consistently over time, whereas the Poisson encoding scatters

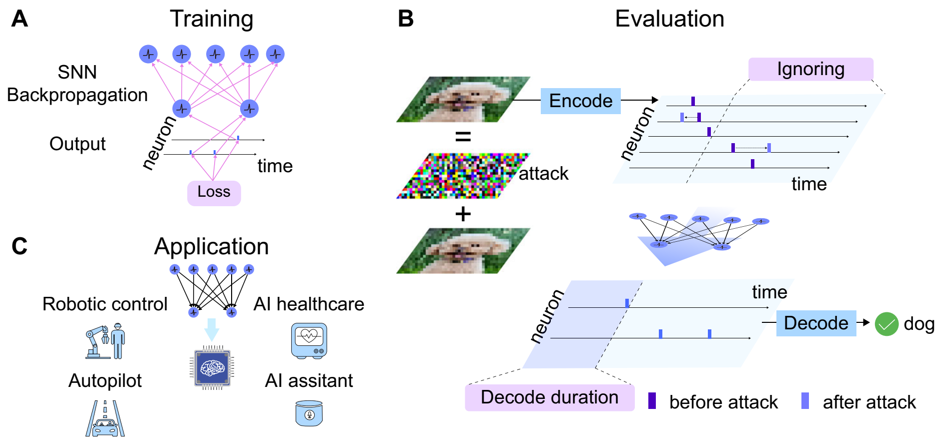

Fig. 1 | SNNs exhibit capacities to protect themselves against adversarial attacks imperceptible to human vision by leveraging meticulously crafted encoding strategies and precise control over decoding duration. A SNNs can be obtained by ANN-to-SNN conversion or directly trained using SNN-specific backpropagation methods. **B** For visual tasks, images are encoded into input sequences that SNNs process to produce spike outputs. Our encoding and decoding approaches prioritize task-critical information during encoding and enable early exits during decoding, thereby mitigating the impact of encoded perturbations. **C** The distinctive robustness of SNNs positions them as a competitive option for safety-critical applications, especially those involving image and sequence data processing, such as in robotic control and autonomous driving.

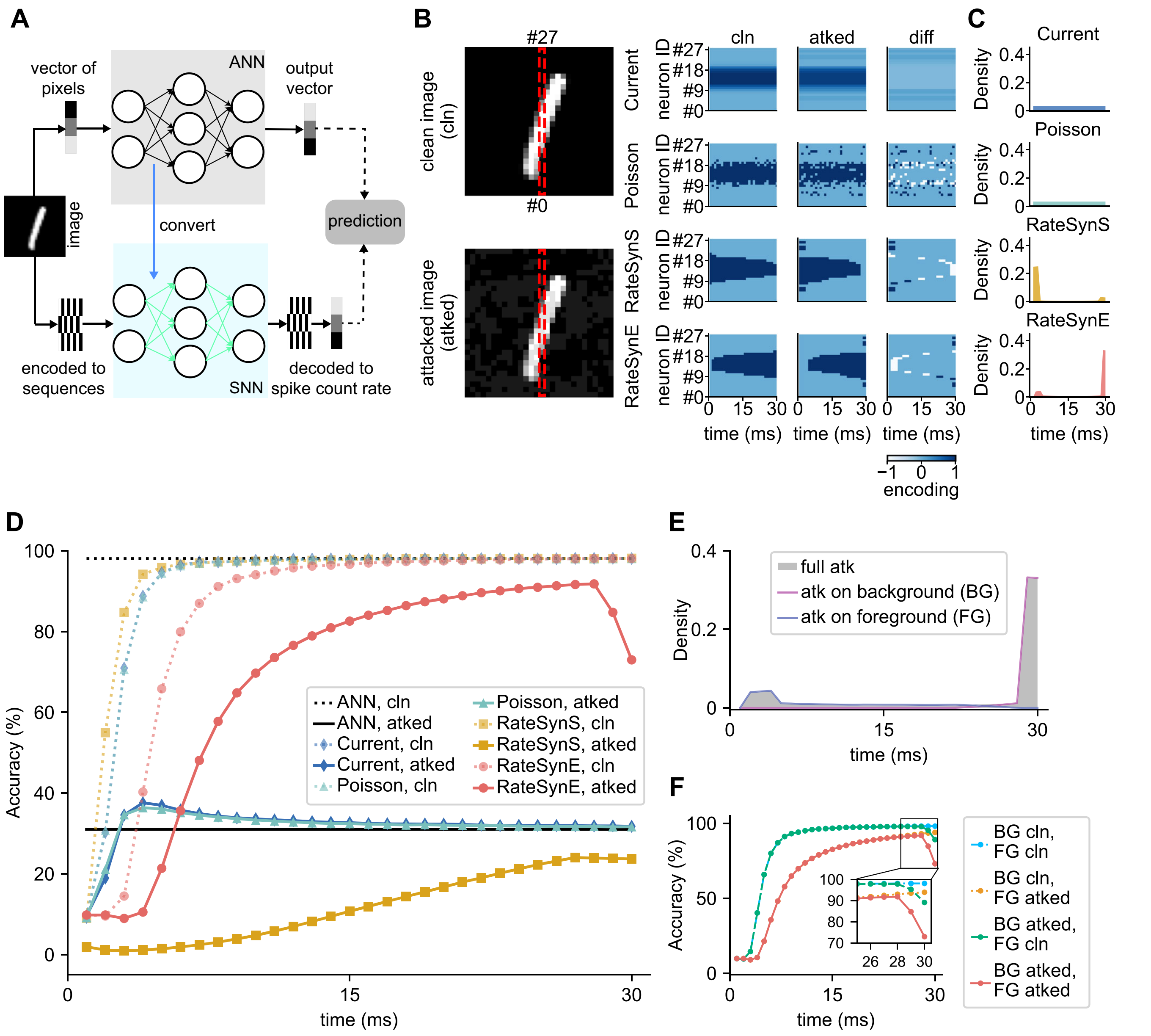

Fig. 2 | Illustration of encoding methods and their impact on SNN robustness. Experiments are conducted on the MNIST dataset for 30 ms. **A** Diagram illustrating the transformation of an ANN into a rate-based SNN for image classification tasks. ANNs process pixel value vectors directly, whereas SNNs utilize encoded sequences as input. The output spike sequences are decoded into spike count rates, which are subsequently translated into predictions. **B** Examples of encoded sequences generated from pixel values on clean (cln) and attacked (atked) images (highlighted by red dashed rectangles) and their differences (diff) using four encoding methods: the current, Poisson, RateSynS (Eq. (9)), and RateSynE (Eq. (10)) encodings. The encoding duration is 30 ms. The current encoding temporally repeats floating-point vectors, distributing perturbations uniformly over time. The Poisson encoding uses Poisson sampling to generate spike trains with probability proportional to the input value. RateSynS and RateSynE, as RateSyn variants, maintain spiking activity over part of the encoding duration, resulting in deterministic perturbation durations. RateSynS aligns spike activity with the start of the encoding duration, while RateSynE aligns it with the end. Supplementary Figs. S1–S3 visualize the temporal mean of the encoded sequences for clean and attacked images using the above encoding methods. **C** Temporal distribution of perturbations for the four encoding methods on the MNIST dataset, illustrating the probabilities of occurrence of encoded value differences over time. **D** Time accumulated accuracy of SNNs using the four encoding methods on clean (cln) and FGSM-attacked (atked) datasets, compared with the accuracy of their counterpart ANN. **E** Temporal distribution of perturbations for RateSynE, displaying contributions from background (BG) and foreground (FG) perturbations under FGSM attack. **F** Time accumulated accuracy with the RateSynE encoding under FGSM attack, evaluated separately for background and foreground perturbations.

them stochastically. The temporal distributions of perturbations in Fig. 2C illustrate the probabilities of encoded value differences occurring within the encoding duration across the entire dataset, which indicates that perturbations are uniformly distributed over time for both current encoding and Poisson encoding.

We evaluate the time accumulated accuracy (TAAcc), which measures classification accuracy over time t, defined as:

$$\mathrm{TAAcc}[t] = \frac{1}{|D|}\sum_{d\in D}\mathbf{1}(\operatorname{argmax}\omega[t] = y_d), \tag{1}$$

where $\omega[t]$ denotes the rate-decoded output vector for the test sample d until time t, y_d is the true label, $\mathbf{1}(\cdot)$ is the indicator function, and $|D|$ represents the dataset size. Figure 2D compares TAAcc curves on clean and attacked datasets, showing similar trends for SNNs with Poisson and current encoding. On clean data, both achieve ANN-equivalent accuracy within 10 ms and remain stable. Under attack, accuracy initially rises to approximately 40% before declining to match the attacked ANN. This suggests converted SNNs offer slight robustness over ANNs during the early simulation phases. To further investigate this phenomenon, we analyze the MNIST images by separating foreground (high pixel values) and background (pixel values close to 0) regions. The full-image perturbations are categorized as affecting either foreground or background regions, and their impacts are isolated in separate experiments (see Supplementary Fig. S4A, B). The results reveal that foreground perturbations cause a slight accuracy drop, while the accuracy of background perturbations follows a trend similar to that of full-image perturbations. This suggests that the early robustness of SNNs arises primarily from their reduced sensitivity to background perturbations, which have a significant impact on overall robustness. When the intensity of background perturbations is set to 0.1, SNN neurons require time to accumulate membrane potentials to reflect the impact of background perturbations. Moreover, higher-intensity attacks result in sharp performance declines (see Supplementary Fig. S4A, B).

The current and Poisson encoding strategies ensure that all pixels contribute to the encoded input sequence at each time, either deterministically or stochastically. The robustness of SNNs with these encoding methods during early simulations suggests that SNNs are inherently robust to mild perturbations due to their integrate-and-fire mechanism. This highlights the potential of SNNs to enhance robustness through temporal processing capabilities, which are not achievable by ANNs. However, performance remains limited when using the current and Poisson encodings. Based on previous observations, achieving improved robustness in SNNs on the MNIST dataset requires an encoding scheme that allows perturbations from different spatial regions to appear at different times. Therefore, we develop a synchronization-based encoding scheme called RateSyn (Eqs. (9), (10)). RateSyn encodes pixel information into spike durations, distributing perturbations throughout the encoding period, with their temporal positions determined by the pixel values (see "Encoding and decoding schemes" section for details). The encoding duration of RateSyn (t_{enc}) matches the SNN simulation duration (t_{sim}). RateSyn maps pixel intensities to spike counts and times within the encoding duration. Specifically, the number of spikes in the encoded sequence is proportional to the corresponding pixel value. Besides, the temporal positions of spike durations are also proportional to the corresponding pixel values. We design two variants of RateSyn: the first variant has spike durations with the same start time (abbreviated as RateSynS, see Eq. (9)), while the second variant has spike durations with the same end time (abbreviated as RateSynE, see Eq. (10)). RateSyn serves as a novel intermediate between rate encodings and temporal encodings. Experiments demonstrate that RateSynE significantly enhances the robustness of SNNs on the MNIST dataset under adversarial attacks.

For the RateSynS encoding, an earlier end time of the spike duration corresponds to a smaller pixel value. Conversely, for the RateSynE encoding, an earlier start time of the spike duration indicates a higher pixel value. As a result, the temporal distribution of encoded perturbations under the RateSynS and RateSynE encodings differs significantly from that observed with the current and Poisson encodings (see Fig. 2B). In Fig. 2C, two distinct peaks emerge in the temporal distributions of perturbations for the MNIST dataset under both RateSynS and RateSynE encodings. For RateSynS, the earlier peaks in the perturbation distribution correspond to background pixels, while the later peaks correspond to foreground pixels. In contrast, for RateSynE, the earlier peaks correspond to foreground, and the later peaks correspond to background pixels, as illustrated in Fig. 2E.

Through RateSyn encoding, the background and foreground perturbations in MNIST images are effectively mapped to distinct durations, contributing to the differing trends observed in the accuracy curves compared to the current and Poisson encodings, as shown in Fig. 2D. Since the spike count rate encoded by RateSynE and RateSynS is proportional to the ANN activation during the simulation time, the SNN accuracy curve on the clean dataset rises to match the accuracy of its corresponding ANN. Due to differences in spike distribution, RateSynS achieves a faster rise, surpassing the performance of the current and Poisson encodings, while RateSynE shows a slower increase. Under attack, the RateSynS curve starts near 0% and does not exceed the attacked accuracy of the ANN. When background perturbations are excluded for RateSynS, its accuracy almost returns to the clean dataset level (see Supplementary Fig. S4), indicating that SNN performance is primarily affected by background perturbations. For the RateSynE encoding, the SNN first receives foreground perturbations, followed by background perturbations. The accuracy curve reflects this sequence, initially remaining flat, then gradually increasing before dropping sharply near the end of simulation time. The inflection points in the accuracy curve (Fig. 2D) strongly correlate with the timing of the foreground and background perturbations (Fig. 2E). When background perturbations are removed, the downward trend at the end of the simulation disappears (Fig. 2F). Conversely, removing foreground perturbations shortens the initial flat phase, where the accuracy does not rise. This suggests that foreground perturbations impede correct classification during early simulation time. As the simulation progresses, before background perturbations take effect, the network performance improves steadily, enhancing robustness. However, once the SNN encounters background perturbations, its performance declines. Low-intensity background perturbations only slightly reduce performance, but as the attack intensity increases, the cumulative effect leads to a sharp decline in performance after exposure to background perturbations (Supplementary Fig. S4C).

Supplementary Fig. S5 shows that the robustness advantage of using RateSynE over current and Poisson encodings for SNNs is maintained even when the input rate is offset. The robustness is further enhanced due to network sparsity induced by the input offset.

Supplementary Fig. S6 illustrates the effect of the encoding methods on spike rates. From the combined results of RateSynE and RateSynS, we conclude that representing foreground perturbations early in the encoded sequence enhances the robustness of SNNs. In contrast, traditional current and Poisson encodings introduce perturbations at any time, either deterministically or stochastically, limiting SNN performance under attack. Additionally, these findings suggest new requirements for SNN decoding methods: if task-critical information is encoded early during the simulation, the decoding method should facilitate early extraction of output information to maximize performance.

Training algorithms for SNNs with flexible encoding achieve robust generalization

Since the robustness of SNNs depends on the timing of information representation, it is preferable to use SNN-specific training algorithms with greater flexibility rather than relying solely on ANN-SNN conversion. While converting an ANN to its SNN counterpart is relatively straightforward, the converted SNN usually requires additional tuning for better generalization. Existing studies have shown that neuron models with leakage factors exhibit higher robustness due to their leakage dynamics, and SNN-specific training algorithms can further enhance robustness[17,22]. Therefore, it is critical to select a flexible SNN training algorithm to improve the utility of temporal information and accommodate richer spiking neuron dynamics. Various direct training methods are available for SNNs[23,24]. This study focuses on two training

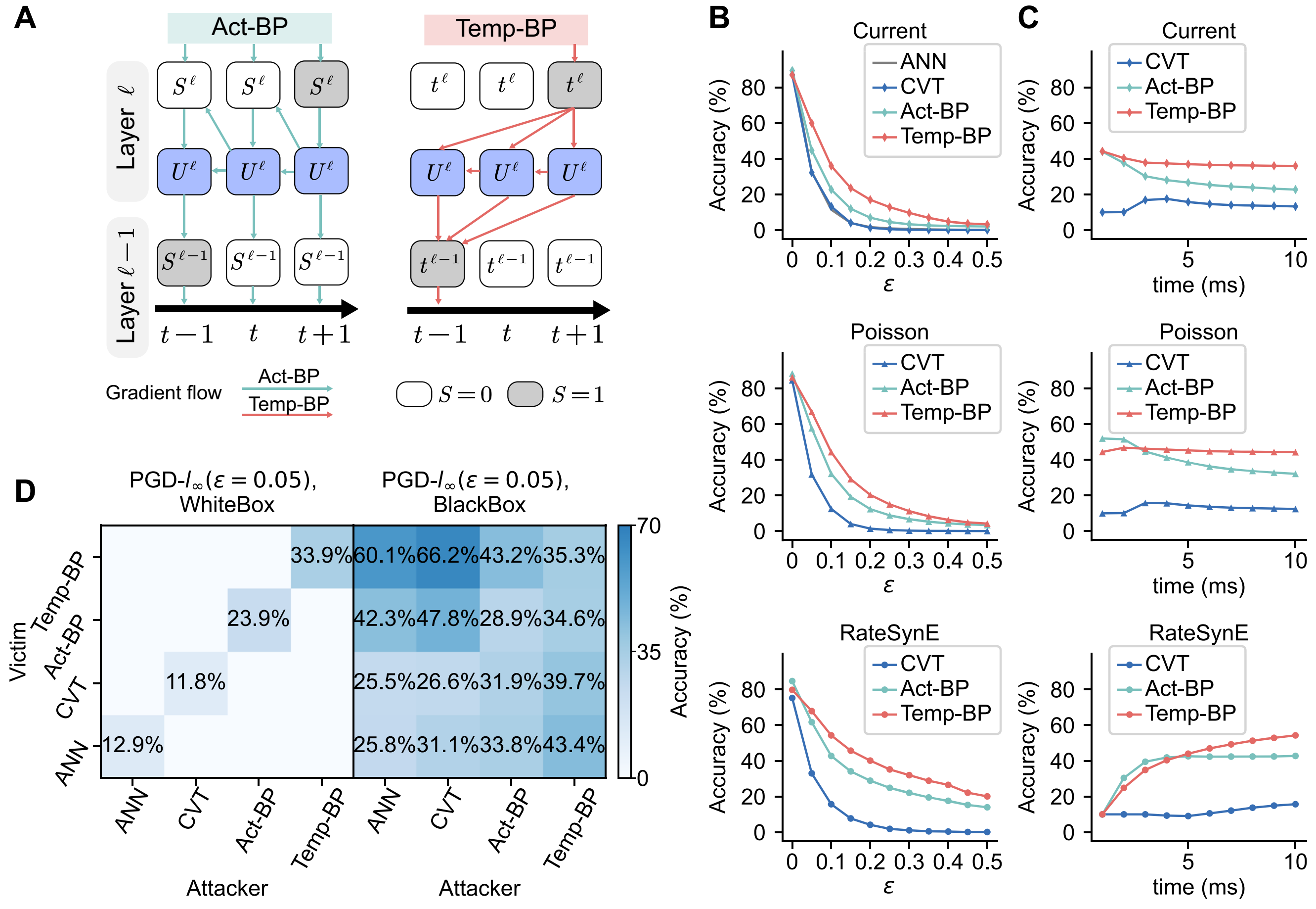

Fig. 3 | Illustration of SNN-specific training methods enhancing SNN robustness. Experiments are conducted on the FashionMNIST dataset for 10 ms. **A** Diagram of the Act-BP and Temp-BP training algorithms. Act-BP back-propagates through membrane potentials over time, while Temp-BP calculates gradients based on spike timing, capturing the difference between expected and actual spikes. **B** Accuracy comparison of converted SNNs (CVT) and SNNs trained with Act-BP and Temp-BP using the current, Poisson, and RateSynE encodings under FGSM attacks with increasing attack intensities ϵ. **C** Time accumulated accuracy of ANNs and SNNs trained with Act-BP and Temp-BP, using the current, Poisson, and RateSynE encodings under FGSM attack ($\epsilon = 0.1$). **D** Robustness of models under white-box and black-box PGD-l_∞ attacks. Darker colors indicate higher post-attack accuracy, with percentages within the grid indicating specific accuracy values.

approaches that leverage the precise timing of spikes: activation-based backpropagation (Act-BP)[25], and temporal-based backpropagation (Temp-BP) methods[26,27]. Act-BP smooths the non-differentiable spike function to implement precise subthreshold membrane potential regulation, whereas Temp-BP directly calculates the derivative of spike timing to achieve accurate spike position adjustment. A schematic diagram of the gradient backpropagation processes for Act-BP and Temp-BP is shown in Fig. 3A. In Act-BP, the sub-gradient of the non-differentiable spike function is determined by the distance between the membrane potential and the firing threshold, regardless of whether a spike occurs. Conversely, Temp-BP disconnects the gradient dependency on membrane potential accumulation when no spikes are generated, allowing it to accurately capture the temporal dependencies for various spike encodings. We implement Act-BP and Temp-BP following the methods proposed by Wu et al.[25] and Zhang & Li[26], respectively (see “Architectures and training approaches” section). Using these algorithms, we train SNNs with the current, Poisson, and RateSynE encodings, which have already demonstrated robustness on black-and-white images. Experiments are conducted on the FashionMNIST dataset, with an encoding duration of 10 ms. The robustness of these models was compared to that of the original ANN and its converted SNN counterpart (CVT) (see “Methods” for details).

We first evaluate the performance of SNNs trained using Temp-BP, Act-BP, and converted SNNs under increasing adversarial attack intensities, as shown in Fig. 3B. Under all three encoding strategies-current, Poisson, and RateSynE-SNNs trained with Temp-BP demonstrate superior robustness compared to those trained with Act-BP, which, in turn, outperform the converted SNNs. For the current and Poisson encodings, the accuracy drops to nearly 0% when the attack intensity reaches $\epsilon = 0.5$. In contrast, SNNs trained with the RateSynE encoding using either Act-BP or Temp-BP maintain accuracy levels around 20%, indicating a synergistic effect between the RateSynE encoding and these training algorithms (Act-BP and Temp-B) in improving robustness. We further examine the TAAcc under a fixed attack intensity ($\epsilon = 0.1$), as shown in Fig. 3C. For the current and Poisson encodings, SNNs trained with Act-BP and converted SNNs exhibit noticeable accuracy drops during the early simulation stage, consistent with patterns observed in Fig. 2D. In contrast, SNNs trained with Temp-BP show minimal early accuracy fluctuations, suggesting that Temp-BP is less sensitive to perturbation-induced variations in spike count rates during early simulation. When using the RateSynE encoding, the accuracy curves for all three approaches rise throughout the simulation. Notably, in the final stage, Temp-BP-trained SNNs achieve the highest post-attack accuracy, outperforming both Act-BP-trained and converted SNNs. These results highlight Temp-BP’s ability to leverage temporal information over extended simulation periods for improved performance.

In addition to providing gradients for SNN training, both Act-BP and Temp-BP also provide gradients for white-box attacks on SNNs. To evaluate white-box attack performance and attack transferability of SNN models and their ANN counterparts, we trained two groups of models, each consisting of two SNNs trained with Act-BP and Temp-BP, a converted SNN, and its ANN counterpart. As shown in Fig. 3D, white-box attack results demonstrate each model's performance under self-attacks, while black-box attack results assess the effectiveness of attacks transferred between groups. In these cases, one model acts as the attacker, and another as the victim. The converted SNN uses Act-BP to backpropagate gradients. Details of the PGD-$l_\infty(\varepsilon = 0.05)$ attack are provided in the "Attack methods" section. The results indicate that naturally differentiable ANNs produce stronger white-box attacks than SNNs trained with Act-BP and Temp-BP. However, SNNs trained with Act-BP and Temp-BP exhibit strong black-box attack capabilities. Notably, the black-box attack launched by SNNs trained with Act-BP or Temp-BP against ANNs is more effective than attacks launched by ANNs against SNNs. Similar patterns are observed for the BIM-$l_2(\varepsilon = 0.5)$ attack in Supplementary Fig. S7.

To further understand the impact of Act-BP and Temp-BP on model robustness, we plot changes in testing loss values under two FGSM adversarial directions, as shown in Supplementary Fig. S8. Supplementary Fig. S9 further supports Supplementary Fig. S8. The loss function of Temp-BP-trained SNNs increases less after perturbation compared to Act-BP-trained SNNs, regardless of the encoding method. This suggests that Temp-BP leads to flatter minima in the loss landscape[28], which correlates with improved robust generalization. Additionally, under the same increase in loss, Temp-BP-trained SNNs require larger perturbations, potentially explaining why attacks generated by Temp-BP-trained SNNs are stronger when using the same attack intensity as Act-BP-trained models.

Early exiting SNN decoding improving robustness by protecting against subsequent perturbations

The priority appearance of task-critical information puts forward requirements for the decoding method of SNN. To enhance SNN robustness, it is crucial to obtain the label as early as possible during the network simulation, thereby preventing further accuracy degradation. This requirement holds even for SNNs that rely exclusively on spike count rate to represent information. By reducing the decoding time, the cumulative effect of perturbations of task-irrelevant information can be minimized, thereby improving accuracy. Based on this, we hypothesize that implementing an early exit decoding strategy is key to enhancing SNN robustness. To test this hypothesis, we employ the Time-to-First-Spike (TTFS) coding strategy, a typical temporal coding approach[29–31]. TTFS coding can encode pixel values into spike sequences and decode spike outputs into output vectors, which are illustrated in Fig. 4A, B. TTFS encoding resembles RateSynE encoding as both prioritize task-critical temporal information by operating over an encoding duration where larger pixel values result in earlier spikes. The key difference lies in TTFS encoding's emission of only a single spike per input.

TTFS decoding determines output predictions based on the timing of the first spike. Differences in spike distribution characteristics can lead to discrepancies between TTFS decoding and rate decoding, which relies on spike count rate, as illustrated in Fig. 4B. The TTFS decoding output is expressed as:

$$\omega_{i,TTFS}(t_{sim}) = \max_{t\in[0,t_{sim}]}((t_{sim} - t)s_i(t)), \tag{2}$$

where $s_i(t)$ is the i-th neuron's spike output, and t_{sim} is the simulation duration. By comparison, rate decoding sums the outputs directly, that is, $\omega_{i,Rate}(t_{sim}) = \int_0^{t_{sim}} s_i(t)dt$. Both the TTFS and rate decodings use the cross-entropy loss function during training, but their gradient sparsities differ. The TTFS decoding computes gradients only at the decoding time, whereas the rate decoding computes gradients at all times. An example of the gradient of the decoded output norm with respect to the spike output is shown in Fig. 4B, illustrating this distinction.

We evaluate the robustness of SNNs using the TTFS and rate decodings under FGSM attacks on the FashionMNIST dataset, as shown in Fig. 4C. SNNs are trained with Act-BP using the current and TTFS encodings. The results show that the TTFS decoding enhances robustness more effectively than the rate decoding, and replacing the current encoding with the TTFS encoding further improves robustness. This suggests that both TTFS decoding and encoding contribute to strengthening robustness. Supplementary Fig. S11 highlights the white-box attack advantages of the TTFS encoding. We also test Temp-BP training with the TTFS encoding. The results show significant performance improvements under strong attack conditions, emphasizing that training methods can further enhance robustness. TTFS decoding provides robust performance across various configurations. This robustness can be attributed to TTFS decoding's shorter effective simulation time. Figure 4D illustrates the temporal distribution of perturbation under the TTFS encoding and the average time of the first spike in the output layer for the FashionMNIST dataset within 10 ms simulation time. TTFS decoding time is approximately half that of the rate decoding. Perturbations occurring after decoding have no impact on classification accuracy.

Since the decoding methods influence the evaluation of the loss, we assess the directional derivatives of six models shown in Fig. 4C. The results indicate that models with smaller directional derivatives demonstrate better robustness. This suggests that robust models exhibit smaller changes in loss value after perturbation. Supplementary Fig. S8 shows that the TTFS decoding consistently reduces loss change after perturbation across all encoding methods, highlighting its general effectiveness for enhancing SNN robustness. Additionally, we observe that the spatial distribution of hidden layer representations trained with the TTFS decoding is more stable before and after perturbations (Supplementary Fig. S10). These findings underscore TTFS decoding's superior robustness compared to rate decoding, particularly when paired with specific encodings and training strategies.

TTFS decoding reduces simulation time, which not only improves robustness but also reduces dynamic energy consumption in SNNs. A key metric for energy consumption is the number of Synaptic Operations (SOPs), as the dynamic energy consumption of neuromorphic hardware is theoretically proportional to SOP[32]. To evaluate SOPs for SNNs before and after the use of TTFS decoding, we use three spike encoding methods-Poisson, RateSynE, and TTFS encoding, as shown in Fig. 4F. The results reveal that TTFS decoding significantly reduces the number of SOPs required for SNN simulation. This reduction is primarily achieved by eliminating the SOPs required for the remaining simulation time after decoding. The SOP count for the RateSynE encoding falls between the counts for the Poisson and TTFS encodings, suggesting that introducing timing in rate encoding without reducing firing rates during the encoding period can still decrease SOPs to some extent. The substantial reduction in SOPs with the TTFS encoding is attributed to its high sparsity during encoding (Fig. 4A) and the reduced spike firing rate in the hidden layers. Supplementary Fig. S12 further illustrates the spike firing rates of the hidden layers for different encoding methods.

SNN employs fused input encoding to enhance performance against complex attacks

In real-world scenarios, various types of attacks exist, including manually designed stickers, graffiti[33], lasers[34], and reflections[35]. Some attacks target specific regions of an image, while others affect only a few pixels. When adversarial perturbations are categorized by the norm constraints used to generate them, common norms such as l_0, l_1, l_2, and l_∞ are often used. To evaluate SNN performance under these

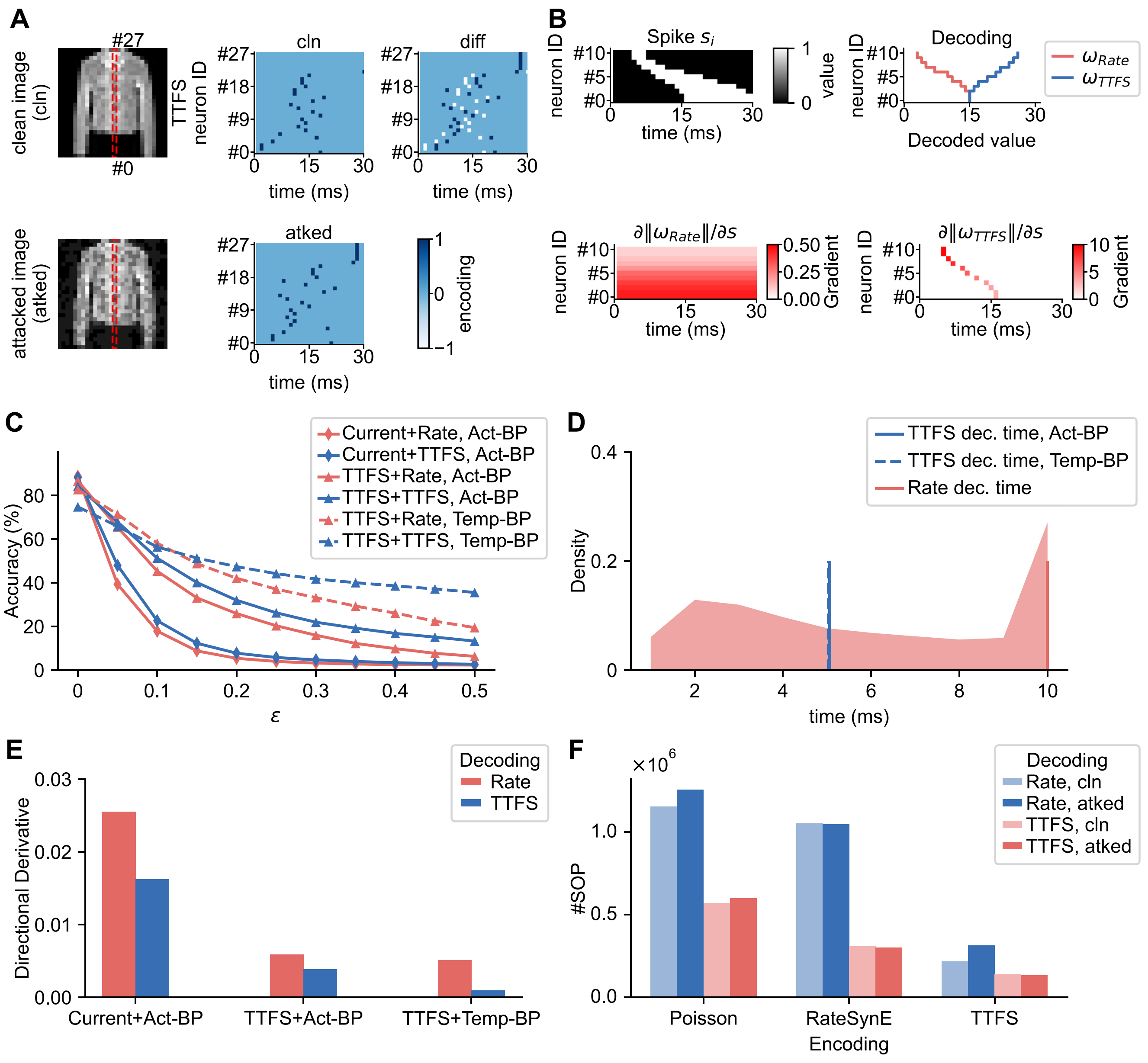

Fig. 4 | Illustration of decoding schemes enhancing SNN robustness. Illustrations for decoding schemes in (**A**) and (**B**) correspond to 30 ms. Experiments are conducted on the FashionMNIST dataset for 10 ms. **A** Diagram of the TTFS encoding, which converts floating-point pixel inputs into spike trains with only a single spike per neuron. **B** Comparison of the rate and TTFS decoding methods. The TTFS decoding (Eq. (2)) assigns values based on the time of the first spike, ignoring subsequent activity and producing a sparse gradient matrix. In comparison, the rate decoding uses spike count rates of the output spike trains as decoded values, incorporating all spikes and resulting in a denser gradient matrix. **C** Accuracy comparison of SNNs using the current or TTFS encodings, paired with either the rate or TTFS decodings, under FGSM attacks. **D** Temporal distribution of perturbations for the TTFS encoding, along with the average time of the first spike in the output layer for the TTFS and rate decoding (dec.) methods. **E** Directional derivatives along the adversarial direction for the TTFS and rate decodings. **F** Number of Synaptic Operations (SOPs) per image for different spike encoding methods combined with the rate or TTFS decodings on clean (cln) and attacked (atked) datasets.

conditions, we use the PGD attack method to generate adversarial images with different norm constraints. The CIFAR-10 dataset, a benchmark in adversarial attack research on SNNs[16,36], serves as the basis for our network training and robustness evaluation. Using ResNet18 as the backbone, we train ReLU-based ANNs and SNNs with different configurations and assess their performance on the attacked CIFAR-10 datasets. Figure 5A presents examples of images attacked with l_0, l_1, l_2, and l_∞ norms. Note that the clean images in Fig. 5A were captured by the authors in natural scenes corresponding to CIFAR-10 categories, and do not originate from the CIFAR-10 dataset. Although adversarially perturbed, these images remain recognizable to the human eye. Under the TTFS encoding, we visualize the temporal distributions of perturbations for the four types of attacks, revealing that norm constraints of the attack impact the temporal distributions of perturbations, which poses challenges for evaluating SNNs' robustness under diverse adversarial conditions.

In addition to evaluating performance on clean and attacked datasets, we also consider prediction stability. Poisson encoding introduces stochastic spikes, leading to variability in label predictions for the same image across trials. To quantify stability, we record the probability of the SNN model predicting the same correct label over 50 trials on a clean dataset. Figure 5C presents the single-trial accuracy and 50-trial stability on clean data, as well as single-trial accuracy on four attacked datasets for SNNs with various input encodings under

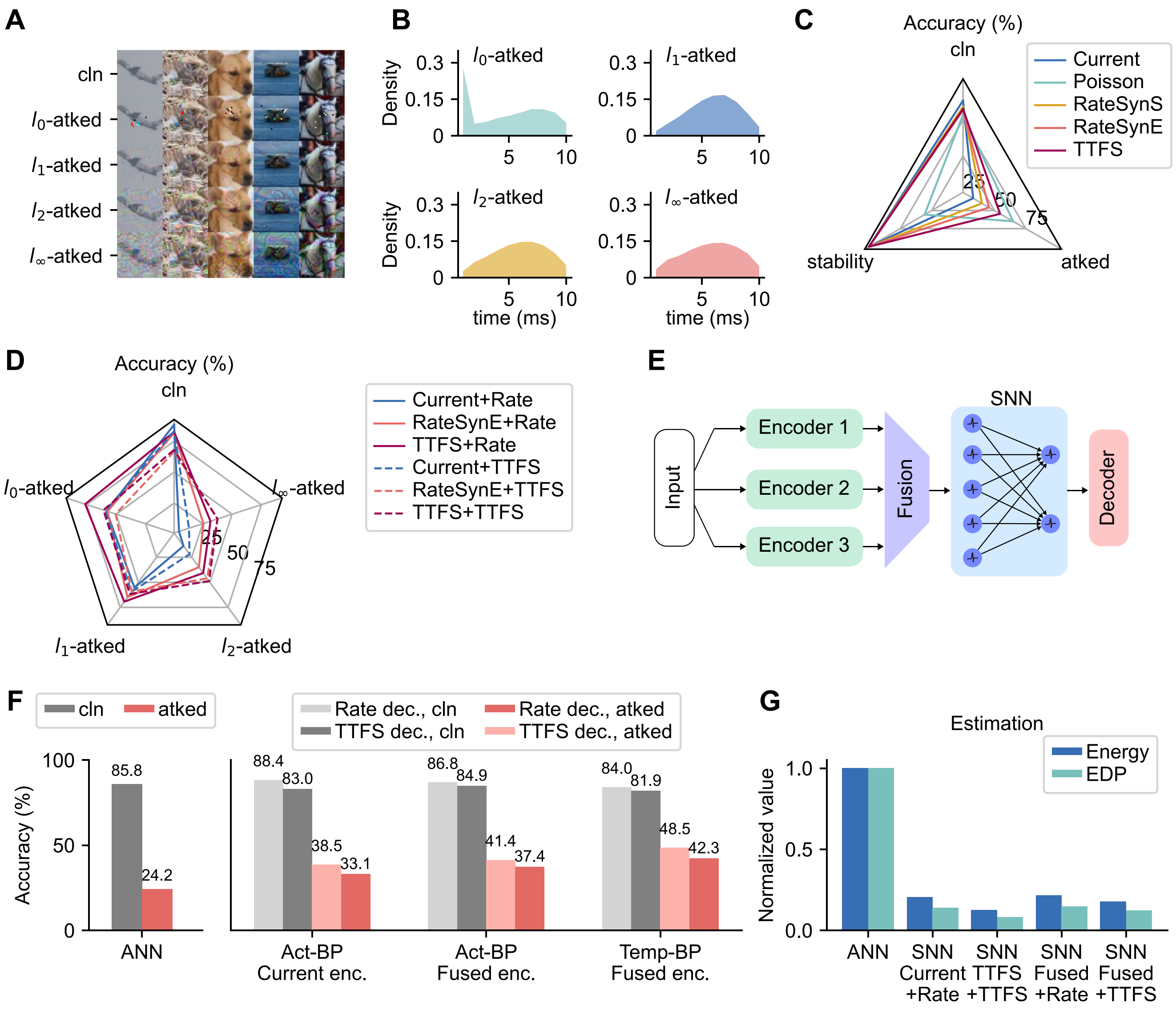

Fig. 5 | Illustration of SNN employing fused encoding to enhance robustness. Experiments are conducted on the CIFAR-10 dataset for 10 ms. **A** Examples of clean images (cln) and adversarial examples (l_0, l_1, l_2, and l_∞-atked). **B** Temporal distributions of perturbations for adversarial examples encoded with TTFS. **C** Radar plot of SNN performance using the rate decoding, showing accuracy on clean data (cln), stability (probability of correct predictions over 50 runs on clean data), and overall accuracy on four types of attacked datasets (atked). **D** Radar plot comparing SNN performance using the rate and TTFS decodings, highlighting accuracy on clean data and four types of attacked datasets. **E** Diagram of an SNN with fused encoding, designed to balance accuracy on clean data and robustness by leveraging multiple encoders. **F** Accuracy comparison of SNNs with the fused encoding (enc.) versus non-fused ANN and SNN models, using the rate and TTFS decodings (dec.) on clean and four attacked datasets. **G** Normalized energy and energy-delay product (EDP) estimations obtained using standard neural network accelerators. See the "Evaluation on energy efficiency" section for the detailed evaluation process.

the rate decoding. The Poisson encoding demonstrates the lowest stability due to its inherent stochasticity. Among the deterministic encodings (current, RateSynE, RateSynS, and TTFS), the current encoding achieves the highest performance on the clean dataset. The TTFS encoding outperforms others on the attacked dataset, followed by RateSynE, despite performing slightly worse than the current encoding on the clean dataset. Figure 5D compares the performance of SNNs using the current, RateSynE, and TTFS encodings, combined with the rate or TTFS decoding, under four types of attacks. The results indicate that SNN models exhibit varying performance depending on the type of attack. To develop an SNN with good performance and robustness against diverse attacks, incorporating multiple encodings is necessary. Since most encodings except the current encoding reduce accuracy on clean datasets, the current encoding emerges as a crucial option for balancing accuracy before and after perturbation.

We propose a fusion encoding strategy to balance the accuracy of SNNs on clean and attacked datasets. The fusion model is shown in Fig. 5E. Image inputs are encoded by multiple encoders, and the resulting outputs are merged along the channel dimension, keeping the time dimension unchanged. Specifically, if an input image has shape (C, H, W), where C, H, W denote the number of channels, height, and width, respectively, then after being processed by a single encoder, the resulting sequence has shape (T, C, H, W), where T is the time step. When fusing K encoders, the outputs are concatenated along the channel dimension, resulting in a merged input of shape $(T, K \times C, H, W)$. Therefore, using fused encoding requires modifying the first convolutional layer to accommodate $K \times C$ input channels instead of C, allowing the network to process all encoded inputs jointly at each time step. The selected decoder then decodes the output sequence of SNN. For CIFAR-10, we select the current, RateSynE, and TTFS encoders for fusion based on their performance. Figure 5F illustrates the impact of the fused encoding and the scalability of our proposed approaches in enhancing the robustness of SNNs on the CIFAR-10 dataset. When trained with Act-BP, SNNs using the fused encoding achieve an average 4.3%performance improvement compared to the current encoding under the rate decoding across all four types of attacks. Switching to the TTFS decoding further enhances performance by 4.0% compared to the rate decoding. When trained with Temp-BP, the performance of both rate and TTFS decoding surpasses that of Act-BP training. Notably, the SNN with the fused encoding, the TTFS decoding, and the Temp-BP training achieves more than double the performance of an ANN with the same architecture. Supplementary Fig. S13 compares the fused SNNs' performance on four types of attacked datasets with that of non-fused models. These findings suggest that fused encoding, paired with effective training and decoding strategies, provides high-performance SNNs with significant robustness improvements across diverse adversarial conditions.

Finally, we analyze the energy consumption and energy-delay product (EDP) of SNNs with the fused encoding compared to ANNs with the same architecture. To estimate energy consumption and EDP, we employ the energy and delay models of hardware accelerators based on the conventional von Neumann architecture from the existing literature[37–39]. For SNNs using the current encoding, or when the current encoding is included in the fused encoding, the energy and delay estimation accounts for the multiply-accumulate operations introduced by the current encoding in the first convolutional layer. As shown in Fig. 5G, SNNs with the TTFS decoding and the fused encoding consume 5.6 times less energy and achieve 8.2 times lower EDP compared to ANNs. Among the fused encoding SNNs, the TTFS decoding demonstrates slightly lower energy consumption and EDP than the rate decoding. Furthermore, compared to SNNs trained with the current encoding, those employing the fused encoding exhibit minimal increases in energy consumption and EDP. Thus, fused encoding enhances robustness without significantly impacting the energy efficiency advantage of SNNs, making it a practical approach for improving performance in energy-constrained systems.

Discussion

In this paper, we systematically demonstrate that precise spike timing plays a pivotal role in enhancing the robustness of neural networks, offering insights into the brain's robustness and paving the way for the development of reliable neuromorphic computing devices. We presented a feasible path for securing artificial intelligence from a neuromorphic computing perspective: achieving robust, brain-inspired deep learning by fully leveraging the temporal processing capabilities of SNNs. Our findings reveal that robustness is closely linked to the timing of information representation and can be significantly enhanced through effective coding strategies. Prioritizing task-critical information within the encoded sequence and reducing exposure time to perturbations with decoding proved to be effective in improving SNN robustness. This improvement became more pronounced by applying SNN-specific training methods designed to enhance generalization in the temporal domain. Additionally, we demonstrated that hybrid coding strategies enhance overall performance under complex attack scenarios for neuromorphic systems.

SNNs are evolved versions of ANNs[40] that naturally extend temporal processing capabilities to neural networks[41,42]. However, these capabilities have not been fully leveraged to enhance robustness. Previous research on SNN robustness has not explicitly addressed the role of temporal capabilities in improving robustness. Sharmin et al. were the first to demonstrate that Poisson coding and the voltage leakage mechanism of neurons can enhance SNN robustness[16]. Additionally, Sharmin et al. showed that training with surrogate gradients improves the robustness of Poisson-coded SNNs[17]. Both studies focused on rate coding and did not directly link SNN robustness to its temporal capabilities. Subsequent studies show that the timing coding of SNNs contributes to robustness but lack analyses of robustness, making it difficult to identify underlying causes[43,44]. More recently, some researchers have suggested that incorporating adversarial samples into the SNN training process can improve robustness. However, these studies primarily use rate coding and fail to highlight the methodological distinctions between SNN and ANN robustness research[36,45]. In contrast, our study explicitly attributes the robustness of SNNs to their unique temporal capabilities, distinguishing them from ANNs. We show that the robustness of SNNs requires the cooperation of encoding, decoding, and training methods.

We designed a concise temporal encoding method named RateSyn to enhance the robustness of SNNs by regulating the spike duration of input sequences. Previous work shows that the robustness gain from spike count rate coding is limited[36,46]. Our experimental results show that the proposed temporal encoding scheme works on an SNN using weights converted from an ANN. When the neural network does not change weights, robustness can be improved by changing the encoding method. Poisson encoding has been widely studied in neuroscience[47]. RateSyn and TTFS encoding are also biologically plausible and share key features with biological neural circuits. There are two commonalities between these two encodings. First, both encodings rely on a global reference time, which can be implemented through synchronization. Synchronization plays a critical role in cortical circuits, particularly in the visual cortex, where precise spike timing is associated with sensory inputs and recurrent oscillations[29]. Such a mechanism enables neurons to establish a shared reference time. Additionally, in the mammalian retina, synchronization and oscillatory modulation are thought to support the integration of local features during early visual encoding[48]. Second, information is encoded in the latency of spikes. Experimental studies show that varying levels of visual activity induce different discharge latencies in primary visual areas[49]. It is also believed that neurons in the retina encode information not only in the discharge frequency but also in the response latency[31]. We demonstrate in Supplementary Fig. S14 that all encoding methods discussed in this work are applicable to sequence data. Additionally, our RateSyn methods can be conceptually related to pulse-width modulation, as shown in Supplementary Fig. S15.

Since we adopted a more complex SNN encoding scheme, the specific training scheme of the SNN needs to be considered to achieve performance improvements. The diverse functionalities of various learning rules for neural systems[50,51] motivate us to use training algorithms that better exploit spike timing and temporal information. The Act-BP and Temp-BP approaches mentioned in this paper are among these algorithms. Act-BP and Temp-BP represent two different types of algorithms. The BP process of Act-BP approximately models the process of spike generation[25,52], while Temp-BP pays more attention to how to use the sparsity of spike timing during training[26]. This work selects a representative from each of the Act-BP and Temp-BP algorithms to explore robustness. Our experiments show that both training types improve SNN robustness, indicating that precise spike encoding requires SNN training methods to effectively utilize temporal information. Furthermore, these training methods have advanced research into specific adversarial attacks on SNNs. In our work, we explore the potential for attacks using heterogeneous training methods, with gradient estimation enabling effective attacks even for RateSyn and TTFS encodings. These findings may inspire further studies on SNN attacks and defenses, addressing current vulnerability challenges for SNN.

Many studies are focused on improving the performance of Act-BP and Temp-BP. For instance, Nowotny et al. proposed adapting various loss functions of Temp-BP for effective training[53], while Gygax et al. explored the theoretical mechanisms behind Act-BP[54]. We believe that implementing both training types on neuromorphic platforms will prominently enhance the robustness of neuromorphic systems. For example, Göltz et al. implemented spike-time gradient and surrogate gradient training algorithms on BrainScaleS-2[55]. Given these ongoing improvements in Act-BP and Temp-BP and corresponding system implementations, we expect the robustness of neuromorphic physical systems to inputs to increase greatly in the future.

Neural decoding is concerned with how neuronal responses are translated into meaningful labels[56]. One common decoding scheme in image recognition is using the spike count rate at output neurons as a vector for model predictions[19]. However, this approach actually overlooks the potential benefit of spike timing. To this end, we integrated the timing-sensitive TTFS decoding scheme into SNNs. TTFS decoding uses the first spike time of output neurons to determine the decoding exit time. Transitioning from rate decoding to TTFS decoding alters the loss landscape, introducing a flatter landscape. This may relate to sparse gradients observed with TTFS. Therefore, our work can inspire the development of more robust decoding strategies.

SNNs are beneficial for implementations on various edge devices with low energy budgets[57–59]. However, these devices are now vulnerable to complex real-world adversarial patches[60], with some attacks succeeding by altering just one pixel[61]. To simulate these complex attack conditions, we used the CIFAR-10 dataset and performed attacks with various norm constraints. Our results show that our suggestions on encoding, decoding, and training methods for improving SNN robustness can generalize to these conditions. Complex neural coding schemes are employed in the visual pathway[62]. In our experiments, SNNs trained with the fused encoding achieve a balance between performance on natural and adversarial data. These findings help clarify the roles of different neural coding schemes in biological systems.

At present, we focus on exploring the robustness of SNNs through their temporal capabilities, without performing specific defense measures designed for adversarial attacks. SNNs are trained only on unattacked datasets and are passively influenced by attacks. In fact, our contribution in this work is orthogonal to the design of adversarial defenses. Our discussion of SNNs primarily focuses on point neurons without considering the effects of complex neuronal structures and dynamics (such as synaptic conduction) on precise spike timing. This limitation arises from the current SNN training techniques, which are primarily designed for point-neuron-based models. In addition, these training algorithms impose constraints on simulation time, meaning that the robustness of SNNs under extended simulation times remains an open question. In addition, we mainly used the black-box gradient attacks derived from ANNs, as these are naturally differentiable, to ensure experimental fairness. The robustness of SNNs under SNN-specific attacks remains to be explored. Our work offers the first systematic investigation of how temporal capabilities enhance SNN robustness, providing a foundation for further research into the roles of encoding, decoding, and learning processes in the next generation of neuromorphic computing models[63].

Methods

Neuronal and synaptic models

Artificial Neural Networks (ANNs) typically consist of multilayered neurons equipped with nonlinear activation functions. Among these, we specifically focus on the ReLU (Rectified Linear Unit) function, detailed as follows:

$$\mathrm{ReLU}(\mathbf{x}) = \max(\mathbf{x}, 0), \tag{3}$$

where $\mathbf{x}$ represents the vector of inputs.

Spiking Neural Networks (SNNs), often considered as the third generation of neural networks, are designed to mimic the dynamic processes of biological neurons. In this study, we utilize both leaky and nonleaky integrate-and-fire models (LIF and IF, respectively) as our primary spiking models[64,65]. For neurons employing these models, the membrane potential $U_i^{(l)}(t)$ of neuron i in layer l at time t evolves according to the following dynamics:

$$\mathrm{IF}\ :\quad R\cdot C\frac{dU_i^{(l)}(t)}{dt} = RI_i^{(l)}(t), \tag{4}$$

$$\mathrm{LIF}\ :\quad \tau_m\frac{dU_i^{(l)}(t)}{dt} = -\left(U_i^{(l)}(t) - U_0\right) + RI_i^{(l)}(t), \tag{5}$$

where τ_m is the membrane time constant, R is the resistance, $I_i^{(l)}(t)$ denotes the input current to neuron i in layer l, and U_0 is the resting potential. C is the capacitance for the IF neuron. Note that Eqs. (4) and (5) can both be generalized into a single form: $\frac{dU_i^{(l)}(t)}{dt} = f(U_i^{(l)}(t), I_i^{(l)}(t))$. Considering the discrete implementation in a computer, the fixed-step first-order Euler method is used to discretize the dynamic functions, and we get

$$\begin{aligned}\Delta U_i^{(l)}[t] &= f(U_i^{(l)}[t], I_i^{(l)}[t])\Delta t,\\ U_i^{(l)}[t+1] &= \Delta U_i^{(l)}[t] + U_i^{(l)}[t].\end{aligned} \tag{6}$$

Here $\Delta U_i^{(l)}[t]$ and $U_i^{(l)}[t]$ represent the discrete form of $\frac{dU_i^{(l)}(t)}{dt}$ and $U_i^{(l)}(t)$, respectively. We set $\Delta t = 1$ ms as the simulation time step to ensure detailed temporal resolution of the neural dynamics. When the membrane potential $U_i^{(l)}[t]$ reaches the threshold U_{th}, the neuron generates an all-or-none spike according to $H(U_i^{(l)}[t] - U_{th})$, where $H(\cdot)$ is the Heaviside step function. Following the spike, the membrane potential is reset to the resting potential U_0. Specifically, at the spike time t, when $U_i^{(l)}[t] > U_{th}$, the potential is reset such that $U_i^{(l)}[t] = U_0$. The emitted spikes can be formed into a spike train$S_i^{(l)}[t] = \sum_k \delta_{t, t_i^{(l),k}}$, where $\delta_{i,j} = [i = j]$ is the Kronecker delta, and $t_i^{(l),k}$ represents the timing of the k-th spike of neuron i in layer l. For the LIF models, we incorporate postsynaptic dynamics $a_i^{(l)}[t]$ that govern interneuron communication between layers, which can be described by $a_i^{(l)}[t] = a_i^{(l)}[t-1] + \Delta t/\tau_s \cdot (S_i^{(l)}[t] - a_i^{(l)}[t-1])$[26], where τ_s is the synaptic time constant. $a_i^{(l)}[t]$ is later used as the input current of layer $l+1$.

The parameters of the neural models used for evaluating MNIST, FashionMNIST, and CIFAR-10 are provided in Table 1. For the MNIST, FashionMNIST, and CIFAR-10 datasets, the simulation times t_{sim} are 30.0 ms, 10.0 ms, and 10.0 ms, respectively, which are discretized into 30, 10, and 10 simulation time steps.

Note that for SNNs converted from their ANN counterparts on the MNIST dataset, we employ the lossless ANN-SNN conversion method proposed by Rueckauer et al.[20] to ensure a fair comparison of performance between ANN and SNN models. This conversion method requires the spiking neurons to exhibit slightly different dynamics from those described earlier. Specifically, we use IF neurons governed by the dynamics in Eq. (4). Following Rueckauer et al.'s recommendations, at the spike time t, when $U_i^{(l)}[t] = U_{th}$, the potential is reset by subtracting the threshold, as expressed by $U_i^{(l)}[t] = U_i^{(l)}[t] - U_{th}$. This reset-by-subtraction mechanism has also been applied in TrueNorth[66] and Loihi[67], demonstrating its effectiveness in achieving high-performance ANN-SNN conversions.

Encoding and decoding schemes

Encoding. We demonstrate that the choice of encoding and decoding methods plays a critical role in determining the robustness of SNNs. The encoding method is responsible for converting floating-point numerical information into a sequence that effectively represents the input data. In this study, we implement and evaluate five input encoding schemes: current encoding, Poisson encoding, RateSynE encoding, RateSynS encoding, and TTFS encoding. The encoding duration is denoted as t_{enc}, which is set to be identical to the simulation time t_{sim}.

Table 1 | Parameters for LIF/IF neuron models

Parameter	MNIST	FashionMNIST	CIFAR-10/ImageNet-100	Description	Unit
R	1.0	1.0	1.0	Resistance	Ω
C	1.0	-	-	Capacitance	mF
τ_m	-	10.0	2.0	Membrane time constant	ms
τ_s	-	10.0	2.0	Synaptic time constant	ms
U_0	-	0.0	0.0	Resting potential	mV
U_{th}	1.0	1.0	1.0	Membrane threshold	mV
t_{sim}	30.0	10.0	10.0	Simulation time	ms
Δt	1.0	1.0	1.0	Simulation time step	ms
T	30	10	10	Number of simulation time steps	

The current encoding refers to a method where floating-point pixel values from images are directly input into an SNN as a continuous electrical current maintained over the encoding duration. This scheme is also known as direct coding[46]. For image-based tasks, assuming that x_i represents the pixel value of the i-th pixel ranging between 0 and 1, the current encoding during the encoding duration t_{enc} can be formulated as:

$$I_{i,Current}(t) = x_i. \tag{7}$$

When considering time-discretized neuron input, the current encoding can be implemented as $I_{i,Current}[t] = x_i$ for $t = 1, 2, \cdots, T$, where T is the number of simulation time steps.

Poisson encoding, a prominent variant of spike count rate encoding, is a widely used rate-based encoding mechanism rooted in neuroscience principles[47]. It is extensively applied in the field of SNNs[3] and has also been applied to facilitate the conversion of ANNs to SNNs[19]. This encoding strategy can be formulated as:

$$I_{i,Poisson}(t) = \text{Bernoulli}(1 - e^{-x_i t}), \tag{8}$$

where Bernoulli represents a random event generator with a probability of occurrence defined by $1 - e^{-x_i t}$. This probability controls the rate of the Poisson process, yielding an estimated rate of x_i. The continuous Poisson encoding process can be discretized by sampling random variables from a uniform distribution $U(0, 1)$ at each time step. If the sampled value is lower than x_i, a spike is triggered. This discrete formulation can be expressed as: $I_{i,Poisson}[t] = \mathbf{1}(r < x_i)$ for $t = 1, 2, \cdots, T$, where r denotes a value sampled from a uniform distribution $U(0, 1)$ and $\mathbf{1}(\cdot)$ is the indicator function.

To enable perturbations from different spatial regions to appear at different times, we introduce the RateSyn (Rate synchronization) encoding method. This approach highlights that the timing of encoding significantly influences the robustness of SNNs. The RateSyn encoding functions over a duration of t_{enc}, similar to rate encoding schemes, where the spike count rate of the generated spike sequence is positively correlated with the pixel value. After the first spike is emitted by RateSyn, subsequent spikes are emitted at equal intervals until the desired spike count rate is achieved. When encoding an entire pixel vector simultaneously, specific synchronization patterns between spike sequences are required. To achieve this, we design two synchronization modes for the RateSyn encoding. One is RateSyn at Start (RateSynS). In this mode, the first spike times of all spike sequences are synchronized to the start of the encoding duration. Another one is RateSyn at End (RateSynE). In this mode, the last spike times of all spike sequences are synchronized to the end of the encoding duration. The RateSynS and RateSynE encoding methods can be expressed in continuous time as follows:

$$I_{i,RateSynS}(t) = h(x_i t_{enc} - t), \quad 0 \le t \le t_{enc}; \tag{9}$$

$$I_{i,RateSynE}(t) = h(t - (1 - x_i)t_{enc}), \quad 0 \le t \le t_{enc}, \tag{10}$$

where $h(t)$ is the spike generation function that generates spikes at equal time intervals. For $h(t)$, a spike is generated when $t \ge 0$, with subsequent spikes occurring at regular intervals, where t is an integer multiple of a fixed interval Δt. The function $h(t)$ can be expressed as $h(t) = \sum_{k=0}^{\infty} \delta_{t,k\Delta t}$, where $\delta_{i,j}$ is the Kronecker delta, and k is a non-negative integer. This formulation ensures spikes are generated at regular intervals of Δt. For discrete time, where Δt corresponds to the simulation time step, the RateSynS and RateSynE encoding methods are discretized as $I_{i,RateSynS}[t] = H(x_i T - t)$; $I_{i,RateSynE}[t] = H(t - (1 - x_i)T)$, where $H(\cdot)$ is the Heaviside step function, and $t = 1, 2, \cdots, T$ represent the discrete time steps in the simulation.

In addition to using spike count rate to encode floating-point information, we also employ a purely temporal encoding method known as time-to-first-spike (TTFS) encoding. TTFS encoding is conceptually similar to RateSynE encoding, as both share the characteristic that the timing of the first spike in the encoded sequence is inversely proportional to the floating-point value. In other words, the larger the floating-point value, the earlier the first spike occurs. The TTFS encoding in continuous time can be expressed as:

$$I_{i,TTFS}(t) = \delta_{t,(1-x_i)t_{enc}}, \; 0 \le t \le t_{enc}, \tag{11}$$

where t_{enc} is the encoding duration. The Kronecker delta $\delta_{t,(1-x_i)t_{enc}}$ ensures that a spike is emitted precisely at $(1 - x_i)t_{enc}$, with no spike at other times. Unlike RateSynE encoding, TTFS encoding generates only one spike. In the discrete-time setting, the TTFS encoding is expressed as $I_{i,TTFS}[t] = 1$, $t = (1 - x_i)T$, where T is the total number of time steps during the encoding process.

Decoding. Decoding methods in SNNs can enhance robustness through early exiting the decoding process. These methods are responsible for transforming time-dependent spike train data into a time-independent output vector. Once the output vector is generated, the index of the neuron with the largest output corresponds to the predicted label. In this study, we explore two decoding methods: spike count rate decoding (rate decoding) and time-to-first-spike (TTFS) decoding. Both TTFS and rate decoding methods employ the cross-entropy loss function during training.

The rate decoding converts spike trains into spike count rates for each sequence. Let $o_i(t)$ denote the output of the i-th neuron at time t.

The *i*-th item in the output vector for the rate decoding is defined as:

$$\omega_{i,Rate}(t_{sim}) = \int_0^{t_{sim}} s_i(t)dt, \tag{12}$$

where t_{sim} represents the total simulation time. In discrete simulations of SNNs, the rate decoding process is discretized as $\omega_{i,Rate}[T] = \sum_{t=1,2,\cdots,T}(s_i[t])$, where T represents the total number of simulation time steps.

TTFS decoding extracts information based on the timing of the first spike in each spike sequence. The *i*-th item in the output vector for TTFS decoding can be expressed as:

$$\omega_{i,TTFS}(t_{sim}) = \max_{t\in[0,t_{sim}]}((t_{sim} - t)s_i(t)). \tag{13}$$

In a discrete form, the above equation becomes $\omega_{i,TTFS}[T] = \max_{t=1,2,\cdots,T}((T+1-t)s_i[t])$.

It is worth noting that we adopt discrete forms of both encoding and decoding in this paper, which are consistent with the discrete-time simulation of spiking neurons.

Architectures and training approaches

In Figs. 2–4, to avoid the robustness influence brought by complex architectures, we conduct our experiments using a fully connected network without bias for both ANNs and SNNs. Our experiments utilize the MNIST and FashionMNIST datasets. The network architecture comprises three feedforward layers, with the number of input neurons set at 784-corresponding to the 28×28 pixels of each image in the datasets. For the MNIST dataset, the number of hidden neurons is 100, while for the FashionMNIST dataset, the number of hidden neurons is 500. The network finally outputs probabilities across 10 classes for classification. In Figs. 2 and 3, we only use rate decoding for SNNs. The cross-entropy loss is used in training for 100 epochs. All neural network training processes are implemented using PyTorch on GPUs.

For the SNNs, we explore three approaches for obtaining the networks: ANN-SNN conversion (CVT), Act-BP, and Temp-BP. The synaptic weights for each layer are denoted as $W_{ij}^{(l)}$, where l represents the layer index, and i and j are the indices of the pre- and postsynaptic neurons, respectively.

In Fig. 2, we employ the lossless ANN-SNN conversion method[20] to obtain a rate-based SNN and evaluate its robustness using various encoding schemes, including current, Poisson, RateSynS, and RateSynE. For the ANNs, we optimize them using backpropagation (BP) with an adaptive moment estimation optimizer with weight decay (AdamW). The decay factor is set to 0.01, and the learning rate is set to 0.001. The core idea behind lossless ANN-SNN conversion is to rescale the weights of the ANN to match the firing rates of IF neurons with the ReLU outputs of artificial neurons. This conversion often employs a reset-by-subtraction mechanism for the IF neurons, and the synaptic kernel function is omitted, allowing neurons to receive direct spike inputs from preceding layers. Following the method proposed by Rueckauer et al., weight rescaling is conducted based on the maximum activations of the layers l and $l-1$ in the ANN, denoted as $\max^{(l)}$ and $\max^{(l-1)}$, respectively. Specifically,

$$W_{ij}^{(l),SNN} = W_{ij}^{(l),ANN} \cdot U_{th} \cdot \frac{\max^{(l-1)}}{\max^{(l)}}, \tag{14}$$

Here, $W^{(l),SNN}$ represents the weight adapted for the converted SNN, and $W^{(l),ANN}$ refers to the weight used in the original ANN. After this conversion, the resulting SNN necessitates a certain period of simulation time to achieve classification accuracy comparable to that of the original ANN. For instance, as demonstrated in Fig. 2D, employing current and Poisson encoding methods requires approximately 5 ms of simulation to achieve ANN-level accuracy.

In Fig. 3, we highlight two training methods designed to improve the robust generalization of SNNs: Activation-based Backpropagation (Act-BP) and Temporal-based Backpropagation (Temp-BP). Act-BP tackles the issue of the non-differentiable spike function by applying a smoothing technique, enabling more effective learning. Temp-BP, on the other hand, facilitates precise adjustments to spike timing. Both methods are utilized to train LIF neurons. The forward pass of layer l in the LIF-based SNN can be formulated as below:

$$U_i^{(l)}[t+1] = \left(1 - \frac{\Delta t}{\tau_m}\right)U_i^{(l)}[t] + \sum_j W_{ij}^{(l)} a_j^{(l-1)}[t], \tag{15}$$

$$S_i^{(l)}[t] = H(U_i^{(l)}[t] - U_{th}), \tag{16}$$

where $H(\cdot)$ is the Heaviside step function. To simplify notation, the a, U, and S in the training method described below are assumed to refer to the same layer by default.

For the Act-BP methods, the core idea is to smooth the Heaviside step function[25]. Following the approach of Neftci et al.[23], we adopt a sigmoid surrogate function to approximate the non-differentiable step function, enabling the computation of gradients for updating model parameters. During the backward pass, the gradient is calculated using the derivative of the sigmoid function, expressed as:

$$\frac{\partial S}{\partial U} = \text{Sigmoid}'(\rho \cdot U), \tag{17}$$

where ρ controls the steepness of the sigmoid function and is set to 5 in this study.

For Temp-BP methods, the key challenge lies in computing $\frac{\partial a}{\partial U}$ based on spike timing. Following the approach by Zhang & Li[26], we separate $\frac{\partial a}{\partial U}$ into interneuron and intraneuron BP components (ϕ_{inter}, ϕ_{intra}):

$$\frac{\partial a[t_k]}{\partial U[t_m]} = \phi_{inter} + \phi_{intra}. \tag{18}$$

The interneuron component arises when the postsynaptic potential is influenced by a presynaptic firing time and is computed as:

$$\phi_{inter} = \frac{\partial a[t_k]}{\partial t_m} \cdot \frac{\partial t_m}{\partial U[t_m]}, \tag{19}$$

where $\frac{\partial a[t_k]}{\partial t_m}$ can be derived by computing $\frac{\partial a[t_k]}{\partial S[t_m]}$ according to Eq. (15) when $S[t_m] = 1$, otherwise $\frac{\partial a[t_k]}{\partial t_m} = 0$. $\frac{\partial t_m}{\partial U[t_m]}$ is computed using the methods proposed by Bohte et al.[68], where the membrane dynamics are approximated as a linear function of t_m, that is, $\frac{\partial t_m}{\partial U[t_m]} = -1/\frac{\partial U[t_m]}{\partial t_m}$. On the other hand, the intraneuron component is defined between an arbitrary time and a presynaptic firing time, which is given by:

$$\phi_{intra} = \frac{\partial a[t_k]}{\partial U\left[t_p\right]} \cdot \frac{\partial U\left[t_p\right]}{\partial t_m} \cdot \frac{\partial t_m}{\partial U\left[t_m\right]}, \tag{20}$$

where $t_m < t_p < t_k$.

We compare the robustness of the LIF-based SNN converted from the ANN against the LIF-based SNNs trained using Act-BP and Temp-BP. To ensure a fair comparison, the same neuronal parameters are used for the LIF-based SNNs trained with Act-BP and Temp-BP, as detailed in Table 1. For the converted SNN, the weights are directly copied from the ANN. This configuration enables the converted network to achieve performance comparable to the ANN on the clean dataset, as shown in Fig. 3B. For the experiments shown in Figs. 3–5, we use a learning rate of 0.0005, the AdamW optimizer, and a weight decay of 0.01.

In Fig. 5, we further demonstrate that SNNs can exhibit robust properties in deeper network architectures. We train the models on the CIFAR-10 dataset, which has been widely used in the previous studies on SNN robustness[16,36,45], highlighting the scalability of our proposed methods to real-world tasks. The ANN employs the ReLU activation function, while the SNN utilizes the neuron parameters listed in Table 1. To enhance model generalization, the images are preprocessed using random cropping and horizontal flipping for data augmentation. The network architecture utilized is ResNet18[69]. During training, we apply a learning rate schedule based on cosine annealing.

Attack methods

Adversarial attacks pose a significant threat to ANNs by inducing inaccurate predictions through small, imperceptible changes to input data. To evaluate model robustness, we employ several widely used attack methods: the Fast Gradient Sign Method (FGSM)[2], Basic Iterative Method (BIM)[70], and Projected Gradient Descent (PGD)[71]. For PGD, we also implement its variants under several norm constraints. All these methods are gradient-based attacks that use the model's gradient to determine perturbations. The stronger the attack magnitude, the greater its effect. These methods are either directly sourced from or adapted from the implementation in the torchattacks package[72]. The attack process involves solving the following constrained optimization problem:

$$\arg\max_{\delta} \mathcal{L}(f(\mathbf{x}+\delta), y) \quad s.t. \quad \|\delta\|_p \le \varepsilon, \tag{21}$$

where δ represents the perturbation applied to the input $\mathbf{x}$, ε is the intensity that constrains the p-norm of δ, $\mathcal{L}$ is the loss function of the network, $\mathbf{x} \in R^{N\times 1}$ is the input image with N pixels, and y is the label of $\mathbf{x}$.

The FGSM is a single-step gradient-based attack that generates perturbations within an l_∞ ball:

$$\delta = \varepsilon\, \mathrm{sign}(\nabla_{\mathbf{x}} \mathcal{L}(f(\mathbf{x}), y)), \tag{22}$$

where $\nabla_{\mathbf{x}} \mathcal{L}(f(\mathbf{x}), y))$ is the gradient of the loss function with respect to $\mathbf{x}$.

Iterative methods, such as BIM and PGD, solve the optimization problem in an iterative manner with a step size of α. For l_∞ iterative attacks, the basic iteration process can be expressed as:

$$\widetilde{\mathbf{x}}^k = \Pi_{l_\infty,\varepsilon}\{\widetilde{\mathbf{x}}^{k-1} + \alpha\, \mathrm{sign}(\nabla_{\mathbf{x}} \mathcal{L}(f(\widetilde{\mathbf{x}}^{k-1}), y))\}, \tag{23}$$

where k denotes iteration steps, and Π denotes the projection of one vector on the l_∞ ball around the vector within the distance of ε. The data at each iteration step is projected back onto the space of the l_∞ ball around the clean data $\mathbf{x}$ with respect to ε. The primary distinction between PGD and BIM lies in the initialization $\widetilde{\mathbf{x}}^0$. BIM uses the raw input as the initial condition, $\widetilde{\mathbf{x}}^0 = \mathbf{x}$, while PGD adds Gaussian noise to the input for initialization. Similarly, for l_2 iterative attacks, the iteration process is expressed as:

$$\widetilde{\mathbf{x}}^k = \Pi_{l_2,\varepsilon}\{\widetilde{\mathbf{x}}^{k-1} + \alpha \frac{\nabla_{\mathbf{x}} \mathcal{L}(f(\widetilde{\mathbf{x}}^{k-1}), y)}{\|\nabla_{\mathbf{x}} \mathcal{L}(f(\widetilde{\mathbf{x}}^{k-1}), y)\|^2}\}, \tag{24}$$

where the Π projection is implemented on the l_2 ball.

For the robustness evaluation shown in Fig. 5, we implement PGD attacks under various norm constraints: l_∞, l_2, l_1, and l_0. The l_1 and l_0 constraints encourage sparsity in the perturbations, as shown in Fig. 5(A). For the l_0-constrained PGD, we adopt the implementation from Croce and Hein[73]. For the l_1-constrained PGD, we use the l_1 projection method proposed by John et al.[74].

When performing attacks, we mainly adopt a black-box attack setting, where attackers have no knowledge about the victim model, considering that SNNs have no natural gradients and that more application scenarios are available for black-box attacks. To obtain the results shown in Figs. 2–5, attacks are performed from ReLU-activated ANNs trained with a different random seed. In this work, we also perform some white-box attacks. Specifically, in Fig. 3D, we use white-box attacks to assess the robustness of different SNN training methods, while in Supplementary Fig. S11, white-box attacks are applied to evaluate various encoding methods. For the current encoding, the gradient with respect to the input can be directly derived from the computational graph. For the Poisson encoding, we employ the straight-through estimator proposed by Sharmin et al.[16] to approximate the gradient with respect to the input. For the RateSynS, RateSynE, and TTFS encodings, we backpropagate the gradient to the spike train and sum the gradients at the time steps where spikes occur to obtain the gradient with respect to the input.

In Fig. 2, we separate the background and foreground regions of the MNIST test images to analyze performance differences under perturbations applied to each part. A threshold of 0.2 is used: pixels with values below this threshold are classified as the background, while those above are classified as the foreground. Using these foreground and background masks, the attacks are divided into two distinct components.

In Figs. 2–4, the reported accuracy represents the average accuracy under attacks from five models. In Fig. 2B and C, an FGSM intensity of 0.1 is used to generate the illustrations and temporal distribution of perturbations. Similarly, in Fig. 2D–F, as well as in Fig. 3C and Fig. 4A, D, and F, the FGSM intensity is also set to 0.1. In Fig. 5, we apply PGD attacks with l_0, l_1, l_2, and l_∞ norms on the CIFAR-10 dataset. For the l_0 attack, the parameter controlling sparsity, k, is set to 10, and the number of PGD iterations is set to 40. For the l_1 attack, the PGD intensity is set to 20, the step size is set to 1.25, and the number of iterations is 40. In the case of the l_2 attack, the intensity is set to 2.5, with a step size of 0.15625 and 40 iterations. Finally, for the l_∞ attack, the intensity is set to 0.1, the step size is set to 0.00625, and the number of iterations is also 40. We also perform evaluations on the ImageNet-100 dataset[75], a subset of ImageNet-1K. Please refer to Supplementary Table S1 for the results. For the l_0 attack, the sparsity parameter k is set to 500, with 40 PGD iterations. For the l_1 attack, the intensity is 1000 and the step size is 62.5, using 40 iterations. Other attack parameters are consistent with those used for CIFAR-10.

Robustness analysis

In this work, we employ various schemes and metrics to evaluate the robustness of neural networks. Key indicators of robustness include classification accuracy under adversarial attacks (Figs. 2–5) and the attack success rate (ASR) (Supplementary Fig. S7). Additionally, we analyze temporal distributions of perturbations and changes in loss value to gain deeper insights into the robustness and generalization capabilities of SNNs.

Classification accuracy under adversarial attacks is defined as the proportion of input data correctly classified by the neural network when subjected to adversarial attacks. For SNNs, the predicted label is determined at the end of the simulation (t_{enc}). The accuracy for SNNs is expressed as:

$$\mathrm{Acc} = \frac{1}{|D|} \sum_{d\in D} \mathbf{1}(\mathrm{argmax}\,\omega[T] = y_d), \tag{25}$$

where $\omega[T]$ is the decoded output vector for test sample d at the end of the simulation, y_d is the true label of d, and $\mathbf{1}(\cdot)$ is the indicator function. $|D|$ represents the total number of samples in the dataset D.

To better understand the temporal processing dynamics of SNNs, we track classification accuracy at each simulation time step. This metric, referred to as the time accumulated accuracy (TAAcc), is defined as:

$$\mathrm{TAAcc}[t] = \frac{1}{|D|}\sum_{d\in D}\mathbf{1}(\mathrm{argmax}\,\omega[t] = y_d). \tag{26}$$

where $\omega[t]$ denotes the decoded output vector at time step t.

In addition, for stochastic encoding methods, such as Poisson encoding, we evaluate the stability of the SNN by measuring its probability of consistently outputting correct labels on a clean dataset across multiple simulations. Stability is defined as:

$$\mathrm{Stability} = \frac{1}{K\cdot|D|}\sum_{k=1,2,\cdots,K}\sum_{d\in D}\mathbf{1}(\mathrm{argmax}\,\omega[T] = y_d), \tag{27}$$

where K is the number of repeated tests.

Attack success rate (ASR) measures the proportion of adversarial samples that successfully mislead the network into producing incorrect predictions. ASR is calculated as:

$$\mathrm{ASR} = \frac{N_{success}}{N_{attack}}\times 100\%, \tag{28}$$

where $N_{success}$ represents the number of adversarial samples that cause misclassification, and N_{attack} represents the total number of testing samples. In Supplementary Fig. S7, ASR is used to evaluate the effectiveness of different attack strategies. A lower ASR indicates stronger robustness of the victim model.

To serve as the input sequence for the SNN, the image must be processed by an encoding method. This allows the perturbation caused by an attack to be reflected in the input sequence. By subtracting the encoded sequences of the clean and attacked images, we can represent the perturbation in the sequence according to a specific encoding scheme. For instance, current encoding expresses perturbations for every pixel at each time, whereas TTFS encoding only expresses perturbations at certain times. We compare the perturbation representation in the sequence after encoding with zero. If the result is non-zero, it indicates that the pixel had been perturbed at that time; otherwise, no perturbation occurred. Finally, we calculate the frequency of perturbation occurrences at different times for each image across the entire dataset, yielding the temporal distributions of perturbations. Let $\zeta[t]$ represent the value of the temporal distribution of perturbations for a given encoding method at time t. The process is formulated as follows: $\zeta[t] = \frac{\sum_{d\in D}\sum_{i\in P}[(\tilde{I}_i[t]-I_i[t])\neq 0]}{\sum_{t=1,2,\cdots,T}\sum_{d\in D}\sum_{i\in P}[(\tilde{I}_i[t]-I_i[t])\neq 0]}$ where d is a test sample from dataset D, i is a pixel in the pixel set P, and $I_i[t]$ and $\tilde{I}_i[t]$ correspond to the encoded values before and after perturbation at time t.

In Supplementary Fig. S6, we further examine the impact of different encoding schemes on SNN activation by observing rate changes in hidden layer neurons. In Supplementary Fig. S6A, for a data sample d in dataset D and its attacked counterpart $\tilde{d}$, we compare the average rates $Rate_{cln}$ and $Rate_{atked}$ before and after the attack. These rates are calculated as follows:

$$\mathrm{Rate}[t] = \frac{\sum_{\tau=1,2,\cdots,t}\sum_{j=1,2,\cdots,N_{hidden}}S_j[\tau]}{t}, \tag{29}$$

where N_{hidden} is the number of hidden layer neurons and $S_j[t]$ is the all-or-none spike emission of the j-th neuron at time t. Additionally, we compare $Rate_{atked}$ with the activation Act_{atked} of the ANN after the attack (Supplementary Fig. S6B). Notably, Act_{atked} is normalized to the interval [0,1] for the comparison with $Rate_{atked}$.

We also examine the firing rate of the hidden layer neuron population at a given time under different attack intensities ϵ. The firing rate, $HiddenRate[t]$ is calculated as:

$$\mathrm{HiddenRate}[t] = \frac{\sum_{j=1,2,\cdots,N_{hidden}}S_j[t]}{N_{hidden}}. \tag{30}$$

Finally, to assess robustness from a training perspective, we analyze the directional derivatives of the SNN before and after the attack. The directional derivative is computed as:

$$\nabla = (\mathcal{L}(f(\mathbf{x}+t\mathbf{v}),y) - \mathcal{L}(f(\mathbf{x}),y))/t, \tag{31}$$

where $\mathcal{L}$ is the loss function, $\mathbf{x}$ is the original data, $\mathbf{v}$ is the FGSM adversarial direction, and y is the label. We set $t = 0.05$. Additionally, we evaluate the loss change under two different FGSM adversarial directions, $\mathbf{v}_1$ and $\mathbf{v}_2$, for varying strength combinations (t_1, t_2). The loss change (LC) is given by:

$$\mathrm{LC}(\mathbf{v}_1,\mathbf{v}_2,t_1,t_2) = \mathcal{L}(f(\mathbf{x}+t_1\mathbf{v}_1+t_2\mathbf{v}_2),y) - \mathcal{L}(f(\mathbf{x}),y), \tag{32}$$

where t_1 and t_2 ranged from [−1,1].

Evaluation of energy efficiency

In Fig. 4, we compare the number of synapse operations (SOPs) in the SNN under different encoding and decoding schemes. Traditional ANNs use floating-point operations (FLOPs) to estimate computational overhead, where FLOPs correspond to MAC (Multiply-Accumulate) operations on hardware. In contrast, the SOP for SNN quantifies the number of synaptic operations performed by all neurons during the simulation. Each SOP corresponds to a more energy-efficient activation-based accumulate (AC) operation[32] on neuromorphic hardware. For the SNN model using the TTFS decoding, the simulation duration only accounts for the time required to decode the label using TTFS. We define the ratio $\xi^{(l)}$ as the proportion of spikes generated by the spiking neurons from the l-th layer across all neurons in the layer during simulation. This ratio is used to calculate SOP as:

$$\mathrm{SOP}^{(l)} = \xi^{(l)}\mathrm{FLOP}^{(l)}, \tag{33}$$

where $\mathrm{SOP}^{(l)}$ and $\mathrm{FLOP}^{(l)}$ represent the number of SOPs required for SNN and the number of FLOPs required for ANN, respectively. In Fig. 4F, we show the total number of SOPs summed across all layers.

In Fig. 5G, we present the energy consumption and energy-delay product (EDP) for both SNN and ANN models, using the same deep network architecture, primarily composed of 2D convolutional layers. The energy consumption and delay models for ANN are based on those used by Ali et al.[38] and Kang et al.[37], while the SNN energy and delay models follows Datta et al.[39] The parameters for these estimation are provided in Table 2 following 32-bit configuration for 65 nm CMOS technology mentioned by Datta et al.[39] For the l-th convolutional layer, let the input channel be $C_{in}^{(l)}$, the output channel be $C_{out}^{(l)}$, the kernel size be $k^{(l)}$, and the output height and width be $H_{out}^{(l)}$ and $W_{out}^{(l)}$, respectively. The energy model for the ANN is:

$$E_{ANN}^{(l)} = C_{in}^{(l)}C_{out}^{(l)}(k^{(l)})^2E_{read} + C_{in}^{(l)}C_{out}^{(l)}(k^{(l)})^2H_{out}^{(l)}W_{out}^{(l)}E_{mac} + P_{leak}T_{ANN}^{(l)}, \tag{34}$$

where the first term represents memory access energy for weights, the second term is the energy required for MAC operations, and the third term accounts for static leakage energy. The processing delay $T_{ANN}^{(l)}$ for

Table 2 | Parameters for energy and delay estimation

Parameter	Value	Description	Unit
B_W	32	Bit width of the weights in SRAM	
B_{IO}	64	Bits fetched from SRAM to the processor per bank	
N_{bank}	4	Number of SRAM banks	
N_{mac}	175	Number of MACs in processing element array	
N_{ac}	175	Number of ACs in processing element array	
E_{read}	5.2	Energy required to fetch data from SRAM to the processor	pJ
T_{read}	4	Time required to fetch data from SRAM to the processor	ns
P_{leak}	2.4	Standby power consumption of SRAM memory	nW
E_{mac}	3.1	Energy required to perform one MAC in the processor	pJ
E_{ac}	0.1	Energy required to perform one AC in the processor	pJ
T_{mac}	4	Time required to perform one MAC in the processor	ns
T_{ac}	0.4	Time required to perform one AC in the processor	ns

the l-th layer in the ANN is:

$$T_{ANN}^{(l)} = \frac{C_{in}^{(l)}C_{out}^{(l)}\left(k^{(l)}\right)^2}{\frac{B_{IO}}{B_W}N_{bank}}T_{read} + \frac{C_{in}^{(l)}C_{out}^{(l)}\left(k^{(l)}\right)^2}{N_{mac}}H_{out}^{(l)}W_{out}^{(l)}T_{mac}. \quad (35)$$

Using $\xi^{(l)}$, we extend the energy models to the SNN:

$$E_{SNN}^{(l)} = C_{in}^{(l)}C_{out}^{(l)}\left(k^{(l)}\right)^2E_{read} + C_{in}^{(l)}C_{out}^{(l)}\left(k^{(l)}\right)^2H_{out}^{(l)}W_{out}^{(l)}\xi^{(l)}E_{ac} + P_{leak}T_{SNN}^{(l)}. \quad (36)$$

The SNN delay model only accounts for time reductions due to the conversion of MAC operations to AC operations, but does not include the impact of pulse sparsity on delay, as this depends on hardware support[39], which gives:

$$T_{SNN}^{(l)} = \frac{C_{in}^{(l)}C_{out}^{(l)}\left(k^{(l)}\right)^2}{\frac{B_{IO}}{B_W}N_{bank}}T_{read} + \frac{C_{in}^{(l)}C_{out}^{(l)}\left(k^{(l)}\right)^2}{N_{ac}}H_{out}^{(l)}W_{out}^{(l)}T_{ac}. \quad (37)$$

After calculating the energy consumption and delay for each layer, the total energy consumption and delay of the network are summed to compute the EDP. For SNNs using the current encoding, the estimation of energy and delay for the first convolutional layer follows the same model as ANNs (Eqs. (34) and (35)), as the encoded input is represented in floating-point format. In the case of fused encodings, to ensure accurate estimation, the total estimation for the first convolutional layer is the sum of the estimation of each encoding in the first convolutional layer.

Data availability

The MNIST dataset is a benchmark dataset of 70,000 images of handwritten digits from 0 to 9, created in 1994. The dataset is available at http://yann.lecun.com/exdb/mnist/. The FashionMNIST dataset is also used as a standard for the image recognition task proposed in 2017. The samples are 28 × 28 grayscale images and are categorized into 10 classes. The dataset is available at https://www.kaggle.com/datasets/zalando-research/fashionmnist. The CIFAR-10 dataset is widely used as a benchmark for image classification tasks. It consists of 60,000 color images, each of size 32 × 32 pixels, divided into 10 classes. Each class contains 6000 images, with 50,000 images used for training and 10,000 for testing. The dataset is available at https://www.cs.toronto.edu/~kriz/cifar.html. Source data related to the figures are provided with this paper.

Code availability

The code used for training and simulating neural networks is available at https://github.com/DingJianhao/SNNEnhancingRobustness.

References

1. Park, H.-J. & Friston, K. Structural and functional brain networks: from connections to cognition. *Science* **342**, 1238411 (2013).
2. Goodfellow, I. J., Shlens, J. & Szegedy, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations* (OpenReview.net, 2015).
3. Maass, W. & Zador, A. Computing and learning with dynamic synapses. *Pulsed Neural Netw.* **6**, 321–336 (1999).
4. Stein, R. B., Gossen, E. R. & Jones, K. E. Neuronal variability: noise or part of the signal? *Nat. Rev. Neurosci.* **6**, 389–397 (2005).
5. Gollisch, T. & Meister, M. Rapid neural coding in the retina with relative spike latencies. *Science* **319**, 1108–1111 (2008).
6. Quiroga, R. Q. & Panzeri, S. Extracting information from neuronal populations: information theory and decoding approaches. *Nat. Rev. Neurosci.* **10**, 173–185 (2009).
7. Li, S., Xiao, Y., Zhou, D. & Cai, D. Causal inference in nonlinear systems: Granger causality versus time-delayed mutual information. *Phys. Rev. E* **97**, 052216 (2018).
8. Friston, K. The free-energy principle: a unified brain theory? *Nat. Rev. Neurosci.* **11**, 127–138 (2010).
9. Friston, K., Harrison, L. & Penny, W. Dynamic causal modelling. *Neuroimage* **19**, 1273–1302 (2003).
10. Zador, A. et al. Catalyzing next-generation artificial intelligence through NeuroAI. *Nat. Commun.* **14**, 1597 (2023).
11. Furber, S. B., Galluppi, F., Temple, S. & Plana, L. A. The SpiNNaker project. *Proc. IEEE* **102**, 652–665 (2014).
12. Deng, L. et al. Tianjic: a unified and scalable chip bridging spike-based and continuous neural computation. *IEEE J. Solid-State Circuits* **55**, 2228–2246 (2020).
13. Yin, B., Corradi, F. & Bohté, S. M. Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nat. Mach. Intell.* **3**, 905–913 (2021).
14. Bellec, G. et al. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun.* **11**, 3625 (2020).
15. Nicola, W. & Clopath, C. Supervised learning in spiking neural networks with FORCE training. *Nat. Commun.* **8**, 2208 (2017).
16. Sharmin, S., Rathi, N., Panda, P. & Roy, K. Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations. In *European Conference on Computer Vision* 399–414 (Springer International Publishing, 2020).
17. Sharmin, S. et al. A comprehensive analysis on adversarial robustness of spiking neural networks. In *International Joint Conference on Neural Networks* 1–8 (IEEE Compute Society, 2019).
18. Adrian, E. D. & Zotterman, Y. The impulses produced by sensory nerve-endings. *J. Physiol.* **61**, 151–171 (1926).
19. Diehl, P. U. et al. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *International Joint Conference on Neural Networks* 1–8 (IEEE Computer Society, 2015).
20. Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M. & Liu, S.-C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* **11**, 682 (2017).
21. Ding, J., Yu, Z., Tian, Y. & Huang, T. Optimal ANN-SNN conversion for fast and accurate inference in deep spiking neural networks. In *International Joint Conferences on Artificial Intelligence*, 2328–2336 (IJCAI.org, 2021).

22. Li, C., Chen, R., Moutafis, C. & Furber, S. B. Robustness to noisy synaptic weights in spiking neural networks. In *International Joint Conference on Neural Networks* 1–8 (IEEE Computer Society, 2020).
23. Neftci, E. O., Mostafa, H. & Zenke, F. Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* **36**, 51–63 (2019).
24. Yin, B., Corradi, F. & Bohte, S. M. Accurate online training of dynamical spiking neural networks through forward propagation through time. *Nat. Mach. Intell.* **5**, 518–527 (2023).
25. Wu, Y., Deng, L., Li, G., Zhu, J. & Shi, L. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* **12**, 331 (2018).
26. Zhang, W. & Li, P. Temporal spike sequence learning via backpropagation for deep spiking neural networks. In *Advances in Neural Information Processing Systems* Vol. 33, 12022–12033 (Curran Associates, Inc., 2020).
27. Kheradpisheh, S. R. & Masquelier, T. Temporal backpropagation for spiking neural networks with one spike per neuron. *Int. J. Neural Syst.* **30**, 2050027 (2020).
28. Stutz, D., Hein, M. & Schiele, B. Relating adversarially robust generalization to flat minima. In *Proc. IEEE/CVF International Conference on Computer Vision* 7807–7817 (IEEE Computer Society, 2021).
29. Tiesinga, P., Fellous, J.-M. & Sejnowski, T. J. Regulation of spike timing in visual cortical circuits. *Nat. Rev. Neurosci.* **9**, 97–107 (2008).
30. Guo, W., Fouda, M. E., Eltawil, A. M. & Salama, K. N. Neural coding in spiking neural networks: a comparative study for robust neuromorphic systems. *Front. Neurosci.* **15**, 638474 (2021).
31. Grimaldi, A. et al. Precise spiking motifs in neurobiological and neuromorphic data. *Brain Sci.* **13**, 68 (2022).
32. Yao, M. et al. Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip. *Nat. Commun.* **15**, 4464 (2024).
33. Eykholt, K. et al. Robust physical-world attacks on deep learning visual classification. In *Proc. IEEE/CVF International Conference on Computer Vision* 1625–1634 (IEEE Computer Society, 2018).
34. Duan, R. et al. Adversarial laser beam: effective physical-world attack to DNNs in a blink. In *Proc. IEEE/CVF International Conference on Computer Vision* 16057–16066 (IEEE Computer Society, 2021).
35. Liu, Y., Ma, X., Bailey, J. & Lu, F. Reflection backdoor: a natural backdoor attack on deep neural networks. In *European Conference on Computer Vision* 182–199 (Springer International Publishing, 2020).
36. Kundu, S., Pedram, M. & Beerel, P. A. Hire-SNN: harnessing the inherent robustness of energy-efficient deep spiking neural networks by training with crafted input noise. In *Proc. IEEE/CVF International Conference on Computer Vision* 5189–5198 (IEEE Computer Society, 2021).
37. Kang, M., Lim, S., Gonugondla, S. & Shanbhag, N. R. An in-memory VLSI architecture for convolutional neural networks. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **8**, 494–505 (2018).
38. Ali, M. et al. Imac: In-memory multi-bit multiplication and accumulation in 6T SRAM array. *IEEE Trans. Circuits Syst. I Regul. Pap.* **67**, 2521–2531 (2020).
39. Datta, G., Kundu, S., Jaiswal, A. R. & Beerel, P. A. ACE-SNN: algorithm-hardware co-design of energy-efficient & low-latency deep spiking neural networks for 3d image recognition. *Front. Neurosci.* **16**, 815258 (2022).
40. Maass, W. Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* **10**, 1659–1671 (1997).
41. Zheng, H. et al. Temporal dendritic heterogeneity incorporated with spiking neural networks for learning multi-timescale dynamics. *Nat. Commun.* **15**, 277 (2024).
42. Duan, Q. et al. Spiking neurons with spatiotemporal dynamics and gain modulation for monolithically integrated memristive neural networks. *Nat. Commun.* **11**, 3399 (2020).
43. Nomura, O., Sakemi, Y., Hosomi, T. & Morie, T. Robustness of spiking neural networks based on time-to-first-spike encoding against adversarial attacks. *IEEE Trans. Circuits Syst. II Express Briefs* **69**, 3640–3644 (2022).
44. Leontev, M., Antonov, D. & Sukhov, S. Robustness of spiking neural networks against adversarial attacks. In *International Conference on Information Technology and Nanotechnology (ITNT)* 1–6 (IEEE, 2021).
45. Ding, J., Bu, T., Yu, Z., Huang, T. & Liu, J. SNN-RAT: Robustness-enhanced spiking neural network through regularized adversarial training. In *Advances in Neural Information Processing Systems* Vol. 35, 24780–24793 (Curran Associates, Inc., 2022).
46. Kim, Y. et al. Rate coding or direct coding: Which one is better for accurate, robust, and energy-efficient spiking neural networks? In *IEEE International Conference on Acoustics, Speech and Signal Processing* 71–75 (IEEE Press, 2022).
47. Amarasingham, A., Chen, T.-L., Geman, S., Harrison, M. T. & Sheinberg, D. L. Spike count reliability and the Poisson hypothesis. *J. Neurosci.* **26**, 801–809 (2006).
48. Neuenschwander, S., Castelo-Branco, M. & Singer, W. Synchronous oscillations in the cat retina. *Vis. Res.* **39**, 2485–2497 (1999).
49. Celebrini, S., Thorpe, S., Trotter, Y. & Imbert, M. Dynamics of orientation coding in area v1 of the awake primate. *Vis. Neurosci.* **10**, 811–825 (1993).
50. Zenke, F., Agnes, E. J. & Gerstner, W. Diverse synaptic plasticity mechanisms orchestrated to form and retrieve memories in spiking neural networks. *Nat. Commun.* **6**, 6922 (2015).
51. Zenke, F., Poole, B. & Ganguli, S. Continual learning through synaptic intelligence. In *Proc. 38th International Conference on Machine Learning* 3987–3995 (PMLR, 2017).
52. Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G. & Roy, K. Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci.* **14**, 119 (2020).
53. Nowotny, T., Turner, J. P. & Knight, J. C. Loss shaping enhances exact gradient learning with eventprop in spiking neural networks. *Neuromorphic Comput. Eng.* **5**, 014001 (2025).
54. Gygax, J. & Zenke, F. Elucidating the theoretical underpinnings of surrogate gradient learning in spiking neural networks. *Neural. Computation.* **37**, 886–925 (2025).
55. Göltz, J. et al. Gradient-based methods for spiking physical systems. Preprint at https://arxiv.org/abs/2309.10823 (2023).
56. Nakai, T. & Nishimoto, S. Representations and decodability of diverse cognitive functions are preserved across the human cortex, cerebellum, and subcortex. *Commun. Biol.* **5**, 1245 (2022).
57. Woźniak, S., Pantazi, A., Bohnstingl, T. & Eleftheriou, E. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nat. Mach. Intell.* **2**, 325–336 (2020).
58. Frenkel, C. Sparsity provides a competitive advantage. *Nat. Mach. Intell.* **3**, 742–743 (2021).
59. Khan, M. M. et al. SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor. In *IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* 2849–2856 (IEEE Computer Society, 2008).
60. Komkov, S. & Petiushko, A. AdvHat: real-world adversarial attack on ArcFace Face ID system. In *2020 25th International Conference on Pattern Recognition (ICPR)*, 819–826 (IEEE Press, 2021).
61. Su, J., Vargas, D. V. & Sakurai, K. One pixel attack for fooling deep neural networks. *IEEE Trans. Evolut. Comput.* **23**, 828–841 (2019).
62. Webster, M. A. Adaptation and visual coding. *J. Vis.* **11**, 3–3 (2011).
63. Zenke, F. et al. Visualizing a joint future of neuroscience and neuromorphic engineering. *Neuron* **109**, 571–575 (2021).
64. Li, S. et al. Dendritic computations captured by an effective point neuron model. *Proc. Natl. Acad. Sci. USA* **116**, 15244–15252 (2019).

65. Li, S. & Wang, X.-J. Hierarchical timescales in the neocortex: mathematical mechanism and biological insights. *Proc. Natl. Acad. Sci. USA* **119**, e2110274119 (2022).
66. Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U. & Neftci, E. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)* 1–8 (IEEE, 2016).
67. Buettner, K. & George, A. D. Heartbeat classification with spiking neural networks on the Loihi neuromorphic processor. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* 138–143 (IEEE, 2021).
68. Bohte, S. M., Kok, J. N. & La Poutré, J. A. Spikeprop: backpropagation for networks of spiking neurons. In *ESANN* Vol. 48, 419–424 (i6doc.com, 2000).
69. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proc. IEEE/CVF International Conference on Computer Vision* 770–778 (IEEE Computer Society, 2016).
70. Kurakin, A., Goodfellow, I. J. & Bengio, S. Adversarial examples in the physical world. In *Artificial intelligence safety and security* 99–112 (Chapman and Hall/CRC, 2018).
71. Madry, A., Makelov, A., Schmidt, L., Tsipras, D. & Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations* (OpenReview.net, 2018).
72. Kim, H. Torchattacks: a Pytorch repository for adversarial attacks. Preprint at https://arxiv.org/abs/2010.01950 (2020).
73. Croce, F. & Hein, M. Sparse and imperceivable adversarial attacks. In *Proc. IEEE/CVF International Conference on Computer Vision* 4724–4732 (IEEE Computer Society, 2019).
74. Duchi, J., Shalev-Shwartz, S., Singer, Y. & Chandra, T. Efficient projections onto the l 1-ball for learning in high dimensions. In *Proc. 38th International Conference on Machine Learning* 272–279 (PMLR, 2008).
75. Yeh, C.-H. & Chen, Y. IN100pytorch: Pytorch implementation: training resnets on imagenet-100. https://github.com/danielchyeh/ImageNet-100-Pytorch (2022).

Acknowledgements
The authors gratefully thank their colleagues from Peking University, including Tong Bu, Yujia Liu, and Xinyu Shi, for their helpful discussions and support for the paper. This work was supported by the National Natural Science Foundation of China (62422601 to Z.Y., 62176003 to Z.Y., 62088102 to T.H.) and Beijing Nova Program (20230484362 to Z.Y., 20240484703 to Z.Y.).

Author contributions
Z.Y. and T.H. led the research and conceived the initial idea for this work. J.D. elaborated on the idea, implemented the methods, conducted the experiments, and drafted the initial version of the paper. Z.Y., J.K.L., and T.H. designed the overall structure of the work. J.D., Z.Y., J.K.L., and T.H. revised the paper.

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/s41467-025-65197-x.

Correspondence and requests for materials should be addressed to Zhaofei Yu.

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