Applied Intelligence (2025) 55:463
https://doi.org/10.1007/s10489-025-06354-z

Precise spiking neurons for fitting any activation function in ANN-to-SNN Conversion

Tianqi Wang[1] · Qianzi Shen[2] · Xuhang Li[1] · Yanting Zhang[1] · Zijian Wang[1] · Cairong Yan[1]

Accepted: 8 February 2025 / Published online: 19 February 2025
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2025

Abstract
Spiking Neural Networks (SNNs) are recognized for their energy efficiency due to spike-based communication. In this regard, the shift towards SNNs is driven by their ability to significantly reduce energy consumption while maintaining the performance of ANNs. Converting Artificial Neural Networks (ANNs) to SNNs is a key research focus, but existing methods often struggle with balancing conversion accuracy and latency, and are typically restricted to ReLU activations. We introduce Precision Spiking (PS) neurons, a novel dynamic spiking neuron model that can precisely fit any activation function by jointly regulating spike timing, reset voltage, and membrane potential threshold. This capability enables exact parameter optimization via iterative methods, achieving low-latency, high-accuracy ANN-to-SNN conversion. Experiments on image classification and natural language processing benchmarks confirm state-of-the-art results, with a maximum conversion loss of 0.55% and up to 0.38% accuracy improvement over the original ANN. To the best of our knowledge, this method offers a significant advancement over existing approaches by achieving high-precision fitting of arbitrary activation functions with low latency and minimal conversion loss, thus considerably expanding the range of feasible ANN-to-SNN conversions.

Keywords Spiking neural network · Spiking neuron model · ANN-to-SNN conversion · Universal activation function conversion · Adaptive iterative adjustment

1 Introduction

Artificial Neural Networks (ANNs) have recently achieved significant advancements in domains such as computer vision and natural language processing. However, their high computational demands restrict their deployment in resource-constrained environments [1]. In contrast, Spiking Neural Networks (SNNs), often recognized as the third generation of neural networks, present a promising alternative by transmitting information via discrete spikes [2]. This event-driven mechanism substantially reduces energy consumption compared to the continuous activations of ANNs. Unlike ANNs, which process data continuously, SNNs operate only when a spike occurs, thereby improving efficiency. Nevertheless, the discrete and temporal nature of spikes complicates training, as traditional backpropagation techniques cannot be directly applied.

To address these challenges, methods such as Spike-Timing Dependent Plasticity (STDP) [3, 4] and surrogate gradient approaches [5] have been investigated. These methods have facilitated direct training of Spiking Neural Networks, leading to notable advancements in performance. However, they frequently encounter difficulties related to gradient propagation and scalability, particularly when applied to large-scale datasets and complex models.

A promising strategy to overcome these issues involves pre-training ANNs and subsequently converting them into

Tianqi Wang and Qianzi Shen contributed equally to this work.

✉ Zijian Wang
wang.zijian@dhu.edu.cn

Tianqi Wang
wangtianqi@mail.dhu.edu.cn

Qianzi Shen
shenqianzi@cmsr.chinamobile.com

Xuhang Li
lixuhang@mail.dhu.edu.cn

Yanting Zhang
ytzhang@dhu.edu.cn

Cairong Yan
cryan@dhu.edu.cn

1 School of Computer Science and Technology, Donghua University, Shanghai, China

2 China Mobile Shanghai Industry Research Institute, Shanghai, China

SNNs. This approach facilitates more stable training and enhanced performance in energy-constrained settings. The conversion method aims to directly utilize the weights from the ANN, requiring little to no additional training, while transforming the network into a more energy-efficient SNN. The most widely used conversion technique leverages the equivalence between integrate-and-fire (IF) neurons and the ReLU activation function [6, 7], where rate coding ensures that the firing rates of IF neurons approximate the output of ReLU activations. However, this also implies that the IF neuron may not be suitable for converting other types of activation functions, due to the limited ability of IF neurons to handle non-monotonic and globally nonlinear functions. Several researchers have made notable contributions to optimizing the ANN-to-SNN conversion process, particularly by modifying the structure of ANNs or adapting spiking neuron models based on this equivalence relationship [8–10]. For instance, [11] and [9] have investigated the sources of conversion loss in rate-based encoding methods and proposed enhancements to mitigate these losses. Additionally, in the context of high-performance models such as Transformers, recent studies have extended ANN-to-SNN conversion techniques, demonstrating their applicability to complex architectures [12, 13].

However, these methods are typically limited to ReLU-based activations, presenting challenges when converting models utilizing alternative activation functions, such as GELU in BERT and SiLU in EfficientNet. In response, novel non-rate-based conversion approaches-such as time-to-first-spike (TTFS) and temporal pattern coding (TPC)-have emerged [14–16]. While these approaches offer potential, they have yet to be extensively applied to non-ReLU activation functions.

Some studies, such as that of [12], have attempted to convert neural networks with non-ReLU activation functions. However, these approaches typically require modifications to the original ANN architecture and necessitate retraining of the customized ANN, which increases the computational cost of conversion. The FS conversion method, proposed by [17], achieves impressive accuracy in fitting certain activation functions by strictly constraining the relevant parameters of spiking neurons so that their outputs closely match those of artificial neurons in ANNs. Nevertheless, the FS coding requires three additional parameters, which must be learned through network training. This training process is often time-consuming, and for complex activation functions, the trained parameters may not yield optimal results.

Further research into related work suggests that the dynamic nature of SNNs may hold even greater potential. In SNNs, the term “dynamic" refers to the temporal variations in neuronal behavior, contrasting with the static nature of neurons in ANNs. Temporal dynamics, particularly Temporal Information Concentration (TIC), play a critical role in task processing, as shown in studies like [18] and [19]. Existing models that incorporate temporal dimensions and dynamic parameter adjustments often lack flexibility, especially when addressing complex tasks or non-ReLU activation functions.

While both direct training of SNNs and ANN-to-SNN conversion have made progress, dynamic performance remains limited when handling complex tasks or non-ReLU functions. Direct training methods, such as those in [8] and [20], struggle with unstable gradients and long training times due to the discrete nature of spiking neurons. Meanwhile, methods like [21] are effective but lack flexibility, and models like [22] suffer from insufficient theoretical support.

In this paper, we propose the Precise Spiking (PS) neuron model, a novel dynamic spiking neuron designed to convert ANNs with arbitrary activation functions into SNNs. Our model addresses a critical gap in existing conversion methods, which are typically confined to ReLU-based activations. As modern architectures, particularly Transformer-based models, rely increasingly on non-ReLU functions like GELU, the inability of traditional methods to handle these functions limits their applicability. The PS neuron model overcomes this limitation, directly and accurately fitting any activation function, including complex ones like GELU, thus broadening the scope of ANN-to-SNN conversion.

A key innovation in our approach is the introduction of a novel iterative parameter adjustment method, which is rarely seen in ANN-to-SNN conversion. This method dynamically fine-tunes neuron parameters-such as reset potential, membrane threshold, and spike weight-to precisely match arbitrary activation functions.

Unlike traditional stepwise approximation methods, which rely on gradually adjusting the input signal during inference, our approach avoids these incremental adjustments by optimizing the neuron parameters prior to inference. This pre-inference optimization significantly enhances both efficiency and accuracy, enabling the PS neuron model to handle a wider range of activation functions without the need for intermediate conversions or complex adjustments during inference.

We also provide comprehensive theoretical analyses, including convergence and stability, ensuring that our approach can achieve high precision, even when dealing with complex activation functions (Fig. 1).

Our model demonstrates state-of-the-art (SOTA) performance, minimizing conversion loss and latency in ANN-to-SNN conversion, and proving particularly effective for tasks involving complex activation functions in models like Transformers. Experimental results validate the robustness and effectiveness of our approach in real-world applications.

The contributions of this paper are listed below:

- The PS neuron is proposed, which accurately fits any activation function, including non-ReLU types such as

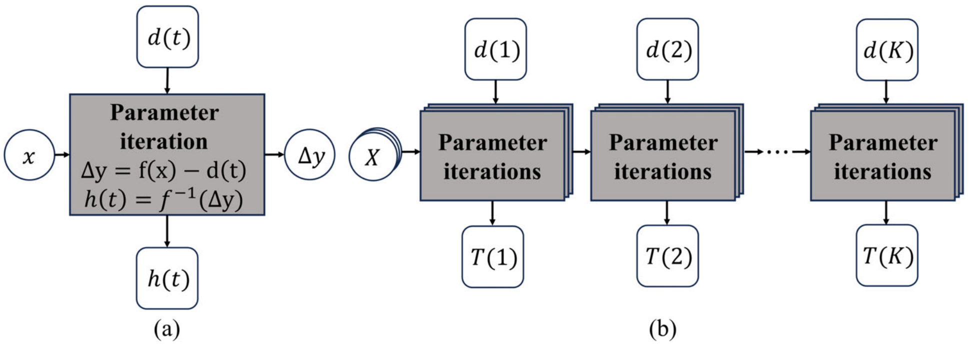

Fig. 1 Parameter iteration process. (a) Each input x is iteratively processed, with Δy used as input for the next timestep. (b) For each input, a reset potential h is generated to determine the membrane threshold T. Spike weight d is predefined

GELU, without requiring additional computation during the inference phase.

- A novel parameter iteration method is introduced, which enhances the PS neuron model's ability to handle a broad range of activation functions efficiently.
- Comprehensive theoretical analyses, including convergence and stability, are provided, along with experimental validation.
- SOTA performance is achieved by our model, minimizing conversion loss and latency in ANN-to-SNN conversion.

In the remaining sections of the paper, Sect. 2 reviews relevant work in the field. Section 3 presents the theoretical foundation of the model we used, the iterative algorithm, and an analysis of the algorithm. Experimental results and their analysis are presented in Sect. 4, followed by discussion and conclution of the findings in Sect. 5.

2 Related work

This section introduces prior work relevant to our research. In ANN-to-SNN conversion studies, many scholars have focused on improving SNN accuracy and reducing inference latency. Constructing efficient spiking neuron models is also a key area of SNN research.

2.1 ANN-to-SNN conversion

The ANN-to-SNN conversion method aims to apply the trained weights of an ANN directly to an SNN, avoiding the challenges of direct SNN training. The most common approach is rate-based encoding, which exploits the equivalence between spike firing rates and the ReLU activation function's output. In this method, ANN activation values are approximated by the firing rate of Poisson spike sequences in the SNN. Higher activation values require more spikes, leading to increased computational cost. To optimize rate-based conversion, researchers have focused on improving accuracy, reducing firing rates, and minimizing latency.

[8] proposed replacing ReLU with a Rate Norm Layer to reduce information loss. [11] and [9] analyzed conversion errors. [11] identified asynchronous SNNs and synchronous ANNs as a key source of errors and introduced the Signed Neuron with Memory. [9] noted the overlooked unevenness error and introduced the quantization clip-floor-shift activation function, achieving zero expected error. [10] proposed the Fast-SNNs model, demonstrating low-latency, high-accuracy performance using quantization and layer-wise fine-tuning.

[12] focused on Transformer-based model conversion with SpikeZIP-TF, enabling the conversion of softmax and layernorm, and surpassing rate encoding's reliance on ReLU. This aligns with [13], who introduced Random Spike Masking for pruning redundant spikes in Transformer SNNs to save energy.

In addition to rate-based encoding, other widely used methods include time-to-first-spike (TTFS) and temporal pattern code (TPC). TTFS encodes information based on the timing of the first spike, typically governed by a global signal. This method often emits only a single spike, resulting in significant energy savings, though at the cost of some accuracy. [14] employed the TPC method, which fully utilizes temporal information by assigning spike weights at different time steps, regulated by a clock signal. This is similar to the phase encoding method proposed by [15], where neuron spikes at different time points are multiplied by corresponding weights to generate varied outputs. [16] also applied temporal coding, minimizing the required time steps using a novel radix operation technique and reducing errors through threshold shifting.

Although the aforementioned studies achieved favorable conversion results, most are limited to ReLU activation functions. [17] introduced the FS conversion method, which enables direct conversion of non-ReLU activation functions. By learning a set of neuron-related parameters, their method ensures that spike outputs reach the activation function's output value within a fixed timestep, thereby completing the conversion process. To accommodate this conversion method, [17] accordingly proposed the FS spiking neuron model.

2.2 Spiking neuron models

Spiking Neural Networks are an abstract model that simulates biological neural systems. In biological neurons, the neuron receives current stimuli, which in turn increase its membrane potential. When the membrane potential reaches a threshold, the neuron releases a spike at a specific time, after which the membrane potential resets to a certain reset potential. Therefore, in the design of most SNN neurons, three key parameters are typically set: membrane potential, reset voltage, and spike timing. Spiking neurons generate sequences of spikes in response to inputs. [7] demonstrated the theoretical equivalence between Integrate-and-Fire (IF) neurons' firing rates and ReLU activation functions, leading to the widespread use of the IF model. This connection has made the IF neuron particularly effective for modeling ReLU, as the firing rate of the IF neuron is highly correlated with the output of the ReLU function. Specifically, the IF neuron fails to reach the threshold when the input is negative, and its output increases linearly with the input when the input is positive, which aligns with the behavior of the ReLU function for $x > 0$. However, this close correspondence between the IF model and ReLU also brings limitations. In particular, it restricts the adaptability of the IF neuron to other types of activation functions, especially those that involve negative values, are nonlinear across all intervals, or are non-monotonic (e.g., GELU), as the IF neuron struggles to accurately represent these functions. [23] proposed Adaptive Spiking Neurons with synchronous spike Sigma-Delta coding, which dynamically adjusts the membrane potential threshold to control spike output rates. [12] introduced the SpikeZIP-TF method for converting Transformer-based models. This approach supports SNN-unfriendly operators such as softmax and layernorm and overcomes the limitation of rate-based encoding, which typically converts only ReLU functions, by replacing the source ANN with a ReLU-based ANN. Among the closely related studies is the Few Spike (FS) neuron model proposed by [17], which enhances adaptability by training parameters to approximate various activation functions. Although FS neurons extend applicability beyond ReLU, they face challenges such as slow parameter training and difficulty achieving optimal parameters for complex functions.

While several other studies have achieved notable results, many have been limited by focusing predominantly on ReLU activations, neglecting the conversion of ANNs with arbitrary activation functions. The limitations of FS neurons, including slow training processes, suboptimal performance, and a limited range of effective input data, are specifically addressed in this research. It is argued that effective conversion from ANNs to SNNs should accommodate arbitrary activation functions. To address this, the Precise Spiking (PS) neuron model is proposed, incorporating a dynamic parameter adjustment method.

2.3 Dynamic of Spiking neuron models

In SNNs, the term “dynamics" refers to the temporal variations of neurons, which contrasts with the static nature of artificial neurons in ANNs. This dynamic characteristic has been explored in various studies, such as in [18], where the temporal dynamics of SNNs were quantitatively analyzed, revealing a concept known as Temporal Information Concentration (TIC).

Recently, several new dynamic neuron models have been proposed to leverage the temporal dynamics of spiking neurons. These models not only incorporate the time dimension into spiking neurons but also allow dynamic adjustments of neuron parameters over time.

Research on dynamic SNNs can generally be divided into two main approaches: direct training of SNNs and first training an ANN before converting it to an SNN. Both approaches have made significant strides but are often constrained by dynamic performance, especially in handling complex tasks or non-ReLU activation functions. In direct training, dynamic membrane potential thresholds have been explored, as seen in [8, 24] and [20], where biologically plausible adjustments were made during training. However, these methods are often limited by the discontinuous nature of spiking neurons, leading to unstable gradients and prolonged training times, which highlights the need for effective ANN-to-SNN conversion methods.

In the field of ANN-to-SNN conversion, stepwise approximation methods are widely used to optimize model performance, reduce energy consumption, and refine network structure. These methods typically involve gradually adjusting network parameters or structures to approach the target optimization state, leading to more accurate conversion and optimization. For instance, [19] proposed the Stepwise Weighted Spiking (SWS) scheme, which introduced a three-value self-resetting neuron model with silent periods to minimize residual errors from stepwise weighting. However, it is limited by its reliance on ANN quantization and manual coefficient settings. [25] introduced a staged DNN-to-SNN conversion strategy, enhancing efficiency through staged conversion and training, though it suffers from limited applicability and heavy computational cost. [26] developed the Hybrid Stepwise Distillation (HSD) method, which optimizes performance at lower time steps, but it is restricted to neuromorphic datasets and incurs substantial computational overhead. [27] presented the Stepwise Leaky ReLU activation function, optimizing membrane potential encoding using a double-threshold dynamic mechanism. While effective for the specific Stepwise Leaky ReLU activation, it does not adapt well to other activation functions.

Although these stepwise methods have made significant contributions, particularly in reducing conversion errors and improving performance, they all require re-calculation and adjustment during the inference stage. This additional computation burden limits their efficiency and increases energy consumption. They also share common limitations, such as reliance on manually set parameters, high computational costs, and limited applicability. These issues hinder the universality and practical use of these methods, emphasizing the need for further improvements. Compared to existing methods, our PS neuron model offers superior flexibility in dynamic parameter adjustment, allowing it to better adapt to complex tasks and non-ReLU activation functions. This flexibility outperforms traditional models in terms of both effectiveness and adaptability.

3 Method

In this section, the PS neuron model is first defined, followed by an introduction to the iterative method employed for the adaptive learning of its parameters. Finally, a detailed explanation of the algorithmic process utilized is provided. The iterative algorithm proposed in this subsection is designed to dynamically generate the model parameters required for PS neuron. This process is carried out offline before the entire SNN model begins inference, and it iteratively generates the parameters dynamically based on the type of activation function used in the SNN. Figure 2 is an example of iteration according to the proposed algorithm, demonstrating how an input value of $x = 2.5$ accumulates spikes to achieve Sigmoid(2.5) using the Sigmoid function as an example.

3.1 Precise Spiking Neuron Model

The PS neuron model is an optimized version of the FS neuron model [17] and primarily consists of three parameters: the membrane threshold T, the spike weight d, and the reset voltage h. The PS neuron differs from the FS neuron in several important ways that significantly enhance its performance. First, the PS neuron utilizes a novel iterative method, allowing it to quickly and accurately identify the optimal parameters for each activation function, unlike the prolonged optimization required by the FS neuron. Additionally, the reset voltage is not a global constant but is tailored to each individual neuron, providing greater flexibility and accuracy. Moreover, the PS neuron can accommodate a wider range of input values, further increasing its adaptability. These distinctions enable the PS neuron to fit a broader range of activation functions with greater speed and precision.

In the PS neuron model, the ith neuron fires at time t only when its membrane potential $v_i(t)$ reaches the threshold $T(t)$. The spiking behavior is described by the Heaviside step function Θ, such that the spike output $z_i(t) \in \{0, 1\}$ is given by (1):

$$z_i(t) = \Theta(v_i(t) - T(t)) \tag{1}$$

After a spike $z_i(t)$ is fired, the neuron's membrane potential is reset to the reset voltage $h_i(t+1)$. Unlike a single scalar value, the reset voltage h_i is a two-dimensional parameter, varying across both neurons and time steps. This is precisely why h_i requires the subscript i to denote the neuron index, while the spike weight $d(t)$ and membrane potential threshold $T(t)$ are shared across all neurons. This feature enables

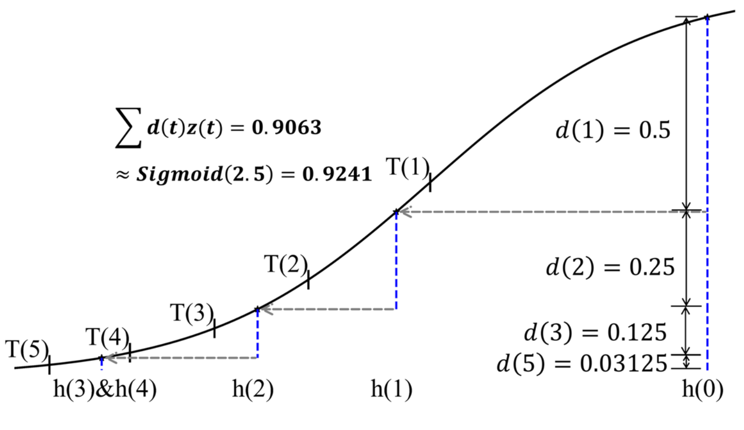

Fig. 2 In the figure, the input value $x = 2.5$ is used as an example. The four blue vertical dashed lines represent the values obtained by the PS neuron at times $t = 1, 2, 3, 5$ for $d(t)$. After each spike emission, the gap between the required value and the target value decreases by $d(t)$. The next time step's corresponding $h(t + 1)$ is found along the gray dashed arrows

the model to determine the most suitable reset value for each distinct input, thereby significantly enhancing adaptability. The dynamic change in the membrane potential of PS neurons can be described by (2):

$$v_i(t+1) = z_i(t)h_i(t+1) + (1 - z_i(t))v_i(t) \tag{2}$$

During the final spike accumulation phase, each spike is scaled by a time-varying weight $d(t)$. The parameter b, a scalar determined by the activation function, represents a vertical shift of the activation value, which will be detailed later. For an activation function output $f(x)$ in ANNs, the PS neuron's output $\hat{f}(x)$, which approximates $f(x)$, is given over K time steps by:

$$\hat{f}(x) = \sum_{t=1}^{K} d(t)z_i(t) + b \tag{3}$$

The three parameters T, h, and d in the PS neuron must be dynamically adjusted through an iterative process based on the input data and the desired activation function to be approximated. Section 3.2 provides a detailed explanation of the iterative process.

3.2 Theorem of parameter iteration process

In the parameter iteration process, to ensure that the final output of the PS neuron matches the output of the Artificial neuron in ANNs, The following theorems are proposed and applied, enabling dynamic adjustment of the PS neuron parameters.

Theorem 1 (Decomposable Activation Theorem) *For a non-negative continuous activation function f that includes zero values, any sufficiently large activation value M can be obtained by summing several distinct smaller activation values.*

Proof Let $M = f(x)$ be a given target activation value and f is a continuous, non-negative activation function that includes zero values. To express M as a sum of smaller activation values, we decompose M into a sum of powers of two, such that:

$$M = 2^m + 2^k + \cdots + 2^s \tag{4}$$

where $m, k, \ldots, s$ are integers with $m > k > \cdots > s$, representing the non-zero bits in the binary (or fractional) representation of M. Let us assume $f^{-1}(0) = a$ and $a < x$, Since f is continuous on the interval $[a, x]$, by the Intermediate Value Theorem, for each power 2^i in the decomposition, there exists some $x_i' \in [a, x]$ such that:

$$f(x_i') = 2^i \quad \text{for} \quad i = m, k, \ldots, s. \tag{5}$$

Therefore, for each of these values 2^i, we can find corresponding inputs x_i' that map to 2^i under the activation function f. By summing these values, we reconstruct the target value M as:

$$f(x) = \sum_i f(x_i') = 2^m + 2^k + \cdots + 2^s = M \tag{6}$$

Hence, M can be represented as the sum of smaller distinct activation values, as required. □

If the activation function f does not include zero or has negative values, its range can be adjusted to start from zero by adding a scalar b. Specifically, let b be the minimum value of f within the given interval. During the iterative process, we can adjust f by subtracting b and then adding b back to the final output. This adjustment ensures that all incremental values, including the smallest ones, are incorporated, thus preventing any gaps in the activation function.

Theorem 2 (SNN Activation Alignment Theorem) *The output of a spiking neuron in an SNN can be precisely matched to the output of an ANN neuron by mapping spike outputs to binary coefficients with weights corresponding to powers of 2.*

Proof To ensure that the output of a spiking neuron in an SNN closely matches the output of an artificial neuron in an ANN, we align the activation values of the ANN with the spike outputs of the SNN. In an ANN, the output $f(x)$ can be expressed as a sum of binary coefficients and powers of 2:

$$f(x) = \sum_{i=s}^{s+K-1} a_i \cdot 2^i \tag{7}$$

where a_i are binary coefficients taking values of either 0 or 1, indicating whether the corresponding term 2^i is included in the sum. The parameter s is an integer that determines the starting index, while K is the number of terms, controlling the range and precision. A smaller s increases precision, and a larger K allows the representation of larger values. Next, we map the spike outputs in the SNN to the activation values in the ANN. We define $d(t) = 2^{s+K-t}$ and $z(t) = a_{s+K-t}$

for $t = 1, 2, \ldots, K$ Then the spiking neuron output for input x, denoted by $\hat{f}(x)$, is represented as:

$$\hat{f}(x) = \sum_{t=1}^{K} z(t) \cdot d(t) = \sum_{t=1}^{K} a_{s+K-t} \cdot 2^{s+K-t} = \sum_{i=s}^{s+K-1} a_i \cdot 2^i = f(x) \tag{8}$$

This mapping ensures that the spiking neuron output $\hat{f}(x)$ matches the ANN output $f(x)$ exactly under fixed precision, such as 32-bit floating point. Therefore, the relationship between spike weights and activation values is maintained with no loss of precision at the specified bit-level representation. □

Theorem 3 *(*Best Parameters for Lossless ANN-to-SNN Conversion*) For an monotonic activation function f and input x, the reset voltage $h(t')$ at any time t' within K time steps should be set to $f^{-1}\left(f(x) - \sum_{t=1}^{t'} z(t)d(t)\right)$ to ensure lossless ANN-to-SNN conversion, where f^{-1} is the inverse function, and K is the total number of time steps required.*

Proof Let the initial membrane potential be $v(0)$, and the output of the PS neuron be $\hat{f}(v(0))$. To ensure that the outputs of the spiking neuron and the artificial neuron match, we assume that $\hat{f} = f$. At time $t = 0$, since no spikes have been fired yet, the cumulative activation value is 0. To make the input x reach the target output, The difference Δy between the current accumulated output value and the target activation value can be expressed as

$$\Delta y(0) = f(x) - 0 = f(x) \tag{9}$$

we set:

$$\hat{f}(v(0)) = \Delta y(0) = f(x) \tag{10}$$

Thus, the initial membrane potential $v(0)$ is given by:

$$v(0) = x = f^{-1}(f(x)) \tag{11}$$

At time $t = t'$, the cumulative activation value is $\sum_{t=1}^{t'} z(t)d(t)$. For an input $x' = v(t')$, we aim to adjust the membrane potential to match the remaining activation value. Thus, we want to satisfy:

$$\Delta y(t') = f(x) - \sum_{t=1}^{t'} z(t)d(t) = f(x') \tag{12}$$

leading to:

$$\hat{f}(v(t')) = \Delta y(t') = f(x') \tag{13}$$

so that the reset voltage becomes:

$$v(t') = x' = f^{-1}\left(f(x) - \sum_{t=1}^{t'} z(t)d(t)\right) \tag{14}$$

This process recursively adjusts the reset voltage at each time step. At time $t'+1$, the new membrane potential is calculated from the inverse function of the remaining activation:

$$\Delta y(t'+1) = \Delta y(t') - d(t'+1)z(t'+1) \tag{15}$$

yielding:

$$v(t'+1) = f^{-1}\left(f(x) - \sum_{t=1}^{t'+1} z(t)d(t)\right) \tag{16}$$

This ensures that, at every time step, the neuron's potential is correctly updated to account for the cumulative activation, allowing the spiking neuron to precisely match the ANN output. □

As illustrated in Fig. 1 (a), Each input x is processed iteratively, with $d(t)$ as input, $h(t)$ generated, and the output of the previous time step $\Delta y = f(x) - d(t)$ used as input for the next step. For non-monotonic functions like SiLU, optimization methods such as Newton's method can approximate x when the inverse is not available. In Sect. 3.3.2, we demonstrate that the computational complexity of using Newton's method is not high and is fully acceptable during model execution.

Based on these principles, a detailed parameter iteration algorithm is designed, described in Section 3.3.

3.3 Parameter optimization: Algorithm and analysis

In this section, we will discuss how to apply the theories introduced in Sect. 3.2 to the dynamic iterative parameters of PS neurons, and analyze their convergence and time complexity.

Algorithm 1 optimizes the parameters h, d, and T by fitting a customizable interval $[l, r]$ with a precision dy, where dy is typically less than 1 (e.g., 0.125). The algorithm is outlined as follows.

3.3.1 Algorithm for iterative parameter optimization

As shown in Fig. 3, the algorithm receives the target activation function f, the fitting interval $[l, r]$, and the desired minimum precision dy, and returns the optimized parameters h, d, T, and b. The algorithm consists of two primary phases: the setup phase, in which necessary values are computed, and the iteration phase, where the spike weights d,

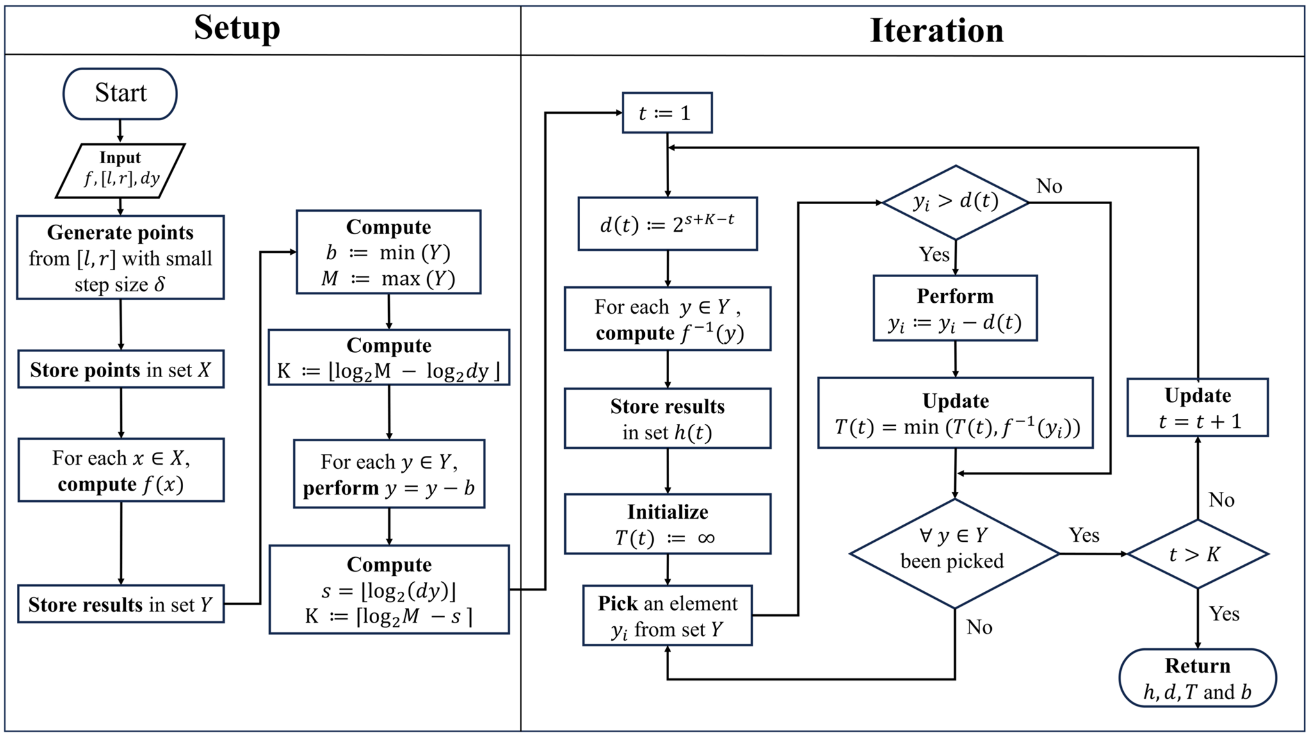
Setup
Iteration
Start
Input $f, [l, r], dy$
Generate points from $[l, r]$ with small step size δ
Store points in set X
For each $x \in X$, compute $f(x)$
Store results in set Y
Compute $b := \min(Y)$ $M := \max(Y)$
Compute $K := \lfloor \log_2 M - \log_2 dy \rfloor$
For each $y \in Y$, perform $y = y - b$
Compute $s = \lfloor \log_2(dy) \rfloor$ $K := \lceil \log_2 M - s \rceil$
$t := 1$
$d(t) := 2^{s+K-t}$
For each $y \in Y$, compute $f^{-1}(y)$
Store results in set $h(t)$
Initialize $T(t) := \infty$
Pick an element y_i from set Y
$y_i > d(t)$
Yes
No
Perform $y_i := y_i - d(t)$
Update $T(t) = \min(T(t), f^{-1}(y_i))$
$\forall y \in Y$ been picked
Yes
No
$t > K$
Yes
No
Update $t = t + 1$
Return h, d, T and b

Fig. 3 Flowchart illustrating the iterative process for optimizing the parameters h, d, and T. The process consists of two main phases: the setup phase, where necessary values are computed, and the iteration phase, where the spike weights d, reset voltage h, and threshold T are adjusted to achieve the desired precision

reset voltage h, and threshold T are adjusted to achieve the desired precision.

Setup Phase

The Setup Phase begins by computing the activation value y for given sample points x.

As illustrated in Fig. 1, individual inputs are denoted by x, a collection of tensor inputs is represented as X.

In the algorithm, the sample points x correspond to the tensor X depicted in Fig. 1(b). Consequently, the results associated with these sample points are also represented as tensors, such as the activation value y.

Next, the minimum value b and the maximum activation value M are determined.The value of M is used to compute the number of time steps K.

According to Theory 2, the output of the SNN is aligned with that of the ANN by defining the weight of the emitted spike as $d(t) = 2^{s+K-t}$, where t is an integer ranging from 1 to K.

For a PS neuron with K time steps, the maximum and minimum values of individual spike weights are 2^{s+K-1} and 2^s, respectively. Therefore, the maximum output value of the SNN is 2^{s+K}, and the minimum precision is 2^s.

To ensure that the required minimum precision satisfies $dy \geq 2^s$, the theoretical constraint for s can be expressed as $s \leq \log_2(dy)$, where s represents the smallest power of 2 and must be an integer. Therefore, s is given by

$$s = \lfloor \log_2(dy) \rfloor \tag{17}$$

Simultaneously, to ensure that the output of the PS neuron reaches the desired maximum value M, the following inequality must hold:

$$2^{s+K} \geq M \tag{18}$$

This implies $K \geq \log_2(M) - s$. Since K must also be an integer, it can be expressed as $K = \lceil \log_2(M) - s \rceil$. Substituting s into this expression yields the final formula for K:

$$K = \lceil \log_2 M - \lfloor \log_2 dy \rfloor \rceil \tag{19}$$

where s is the starting index of the binary representation. Finally, the activation value y is adjusted by subtracting b to align with the conditions specified in the theorem 1.

Interation Phase

The Iteration Phase is responsible for determining the optimal parameters h, d, and T.The spike weight d is defined as shown in Theorem 2, where $d(t)$ can be directly set to 2^{s+K-t}. Here, s represents the smallest power of 2 and is

Algorithm 1 Algorithm Based on Flowchart.

```
1: Input: $f$, $[L, r]$, $\delta$, $dy$
2: Setup Phase:
3: $X \leftarrow$ Generate points in range $[L, r]$ with step size $\delta$
4: $Y \leftarrow \{\}$
5: for each $x \in X$ do
6:     $Y \leftarrow Y \cup \{f(x)\}$
7: end for
8: $b \leftarrow \min(Y)$, $M \leftarrow \max(Y)$
9: $K \leftarrow \lfloor \log_2 M - \log_2 dy \rfloor$
10: for each $y \in Y$ do
11:     $y \leftarrow y - b$
12: end for
13: Iteration Phase:
14: $t \leftarrow 1$
15: while $t \leq K$ do
16:     $d(t) \leftarrow 2^{\lceil \log_2 M \rceil - t}$
17:     $h(t) \leftarrow \{\}$
18:     for each $y \in Y$ do
19:         $h(t) \leftarrow h(t) \cup \{f^{-1}(y)\}$
20:     end for
21:     $T(t) \leftarrow \infty$
22:     for each $y \in Y$ do
23:         if $y > d(t)$ then
24:             $y \leftarrow y - d(t)$
25:             $T(t) \leftarrow \min(T(t), f^{-1}(y))$
26:         end if
27:     end for
28:     $t \leftarrow t + 1$
29: end while
30: Output: $h, d, T, b$
```

determined by (17), while K denotes the number of time steps required to achieve the desired precision, as given by (19).

For the reset voltage h, its configuration relies on the result from Theorem 3. Specifically, at time step t', the reset voltage should be set to the difference δy between the accumulated emitted value and the target output value. This difference corresponds to the inverse of the activation function for the given value. Note that for certain non-monotonic activation functions, the inverse function may not have an explicit analytical solution. In such cases, numerical iterative methods, such as Newton's method, can be employed to find the value of y that satisfies $f(y) = x$.

During the iterative process from $t = 1$ to K, the value of y obtained in the setup phase is progressively reduced by a spike weight at each time step when $y > d(t)$. This ensures that the value of y at time step t equals Δy. Consequently, the reset voltage $h(t)$ at time step t can be calculated straightforwardly.

In Theory 2, the alignment of the spike output $z(t)$ is achieved through the binary representation of the real-valued activation output from the ANN. At each time step t, the threshold $T(t)$ is determined by identifying all input values x for which the current y contributes a binary 1 at that time step. Among these values, the smallest is selected as the threshold $T(t)$. i.e.,

$$T(t) = \min\{f^{-1}(y(t)) \mid y(t) > d(t)\} \tag{20}$$

3.3.2 Convergence and time complexity analysis

Convergence Analysis

Consider an input x, where the output of the artificial neuron in the ANN is denoted by $f(x)$, and the output of the spiking neuron in the PS Neuron is denoted by $\hat{f}(x)$. The difference Δy between these outputs at time step t is defined by (12).

From Theorem 2, it is known that:

$$\sum_{t=1}^{t'} z(t)d(t) \leq f(x), t' = 1, 2, \ldots, K \tag{21}$$

which implies that $\Delta y(t) > 0$.

To analyze the convergence and stability of this iterative algorithm, the Lyapunov function is defined as:

$$\epsilon(t) = \frac{1}{2}|\Delta y(t)|^2 \tag{22}$$

where $\epsilon(t)$ represents the error at the t-th time step (i.e., the t-th iteration). Specifically, the square of the difference between the accumulated value of the current spike emission and the target accumulated value is quantified, divided by 2.

Next, the rate of change of the Lyapunov function is computed. Its derivative is given by:

$$\frac{d(\epsilon(t))}{dt} = \Delta y(t) \cdot \frac{d(\Delta y(t))}{dt} \tag{23}$$

According to (15), the rate of change of the error is:

$$\frac{d(\Delta y(t))}{dt} = -d(t) \cdot z(t) \tag{24}$$

where $d(t)$ is the spike weight and $z(t) \in \{0, 1\}$ is the binary output of the spiking neuron. Therefore, the rate of change of the Lyapunov function is:

$$\frac{d\epsilon(t)}{dt} = -\Delta y(t) \cdot (d(t) \cdot z(t)) \leq 0 \tag{25}$$

Since $\Delta y(t) \geq 0$, $d(t) > 0$, and $z(t) \in \{0, 1\}$, it is concluded that the rate of change of the Lyapunov function is always less than or equal to zero. This implies that the Lyapunov function is non-increasing with each iteration.

Since $d(t)$ decays exponentially, specifically $d(t) = 2^{s+K-t}$, the error $\Delta y(t)$ gradually diminishes over time, with the Lyapunov function approaching zero. As a result, the error decays exponentially, and the system stabilizes, indicating that the algorithm is asymptotically stable and the output of the pulse neuron converges to the target output. Thus, the algorithm is both convergent and stable.

Time Complexity Analysis

The time complexity is divided into two parts: the Setup phase and the Iteration phase.

In the Setup phase, samples are taken in the interval $[l, r]$ with a step size of δ, resulting in $n = \left\lfloor \frac{r-l}{\delta} \right\rfloor + 1$ sample points. For these sample points, the calculations of $f(x)$, the determination of the maximum and minimum values, and the translation operation $y - b$ all have a time complexity of $O(n)$.

In the Iteration phase, K iterations are performed. In each iteration, the inverse function is computed for n sample points to obtain $h(t)$. For non-monotonic functions, Newton's method is employed to find the value of x such that $f(x) = y$. Since Newton's method exhibits quadratic convergence, its computational time complexity is typically very small. Additionally, as the target activation function is usually smooth and differentiable, Newton's method can be approximated as having a time complexity of $O(1)$. Therefore, the time complexity for each iteration is $O(n)$.

In summary, the overall time complexity of the algorithm is $O(Kn)$, where K is the number of iterations. As the error decreases exponentially with the increase in the number of iterations, indicating that K can generally be small. In our experiments, satisfactory results were achieved with K values less than 16, which can be considered a constant. Hence, the total time complexity can be approximated as $O(n)$, where n is the number of sample points in the fitting interval.

4 Experiments

Our proposed PS Neuron is implemented within the PyTorch 2.3.1 framework. The PS neuron model is specifically focused on the conversion of activation functions. Therefore, in all the models used in our experiments, the activation functions are the primary targets of conversion, while other model components, such as the softmax layer, fall outside the scope of our discussion. For image classification tasks, we se-lected the CIFAR10 [28], CIFAR100 [28], and ImageNet [29] datasets, using small-scale ResNet20 [30], large-scale VGG16 [31],and Efficient-Netb7 [32] as the network architectures for the ANNs, respectively. In natural language processing tasks, commonly used datasets include SST-2 [33], ST-5 [33], and CoLA [34], with model selections including the smaller BERT-base [35] and its variant Roberta-base [36], as well as the larger Roberta-large. These models are sourced from public pre-trained model repositories, demonstrating the effectiveness of our con-version method, which requires no prior modification or re-training of the ANN. In model evaluation, we focus primarily on the change in accuracy of the converted SNN compared to the original ANN rather than the absolute accuracy, as absolute accu-racy is directly related to the performance of the converted ANN. The selected network models primarily use ReLU, SiLU, and GELU activation functions. Achieving both lower latency and minimal conversion loss simultaneously is a challenging task. As a result, some works in the table show low conversion loss similar to ours, but they typically require more time steps. Conversely, works with time steps comparable to ours often incur higher conversion losses.

4.1 Accuracy on image datasets

Table 1 presents the experimental results on the CIFAR10 and CIFAR100 datasets. Compared to ImageNet, these datasets are smaller, yet our spiking neuron model achieved a conversion loss as low as 0.07 with only 16 time steps. Competitive methods, such as [37] and [38], also demonstrated strong performance but required significantly more time steps. Other methods, including [9] and [16], achieved relatively low conversion loss with similar time steps but still exhibited slightly higher loss compared to our model.

Table 2 illustrates the performance of our model on the ImageNet dataset. In addition to maintaining the aforementioned advantages, when applied to the EfficientNet-b7 model using the SiLU activation function, our model reduced conversion loss by 1.03% and decreased the number of time steps by three compared to the FS neuron model [17].

The work by [41] employs the ViT-B/32 model, while [42] adopts the EVA model, both based on the Transformer architecture and achieving remarkable results. Despite their difference from more widely used base ANN models such as resnet20, they still provide valuable insights for comparing the conversion losses from ANN to SNN.

In transforming Transformer-based models, both studies require replacing the activation functions in the original ANN with ReLU, followed by retraining and conversion, which incurs unnecessary overhead and additional conversion losses-something our proposed PS neurons do not require. Therefore, the comparative results indicate that PS neurons still hold certain advantages in this context.

4.2 Accuracy on language datasets

PS neurons possess the notable capability of converting any activation function, not limited to ReLU. In BERT models, which conventionally utilize the GELU activation function, PS conversion can be directly applied to transform GELU into a spiking form. We implemented PS con-version on the

Table 1 Comparison of experimental results on CIFAR10, CIFAR100, and ImageNet using various methods

Method	Net. Arch.	ANN Acc.	SNN Acc.	Loss	K
CIFAR10					
[9]	VGG16	95.52	95.40	−0.12	16
[16]	VGG16	93.90	93.76	−0.14	16
[39]	VGG16	95.74	95.58	−0.16	32
[40]	VGG16	92.81	91.13	−1.68	100
[8]	VGG16	92.84	92.51	−0.33	128
[37]	VGG16	93.63	93.63	0.00	2048
[17]	ResNet20	91.58	91.45	−0.13	10
[16]	ResNet20	92.67	92.61	−0.06	16
[39]	ResNet20	96.56	96.11	−0.45	32
[40]	ResNet20	93.15	92.22	−0.93	250
[37]	ResNet20	91.47	91.36	−0.11	1024
[38]	ResNet20	91.47	91.36	−0.11	2048
This work	**VGG16**	**94.15**	**94.14**	**−0.01**	**16**
This work	**ResNet20**	**92.59**	**92.54**	**−0.05**	**16**
CIFAR100					
[9]	VGG16	76.28	76.24	−0.04	16
[16]	VGG16	72.14	71.85	−0.29	16
[39]	VGG16	78.49	74.98	−3.51	32
[11]	VGG16	74.13	73.69	−0.44	64
[37]	VGG16	71.22	70.97	−0.25	2048
[38]	VGG16	71.22	70.93	−0.29	2048
[16]	ResNet20	68.87	68.08	−0.79	16
[9]	ResNet20	69.94	67.33	−2.61	16
[39]	ResNet20	80.69	79.83	−0.86	64
[38]	ResNet20	68.72	67.82	−0.90	2048
[37]	ResNet20	68.72	68.18	−0.54	2048
[41]	ViT-B/32	87.35	85.98	−1.37	256
This work	**VGG16**	**74.00**	**73.96**	**−0.04**	**16**
This work	**ResNet20**	**68.81**	**68.74**	**−0.07**	**16**

The table is divided into two parts, comparing the performance of different models on the CIFAR-10 and CIFAR-100 datasets. Our work is presented in the last row and highlighted in bold. “Loss” represents the difference in accuracy between the SNN and the ANN, i.e., $\mathrm{ACC}_{\mathrm{SNN}} - \mathrm{ACC}_{\mathrm{ANN}}$. “K” indicates the number of time steps used in SNN simulations

BERT model and its variants, with the results presented in Table 3. In the experiments, the parameters used by the FS neurons [17] for the GELU activation function were trained using the code provided in the original paper. While [12] this method achieves some effectiveness in non-ReLU models, our proposed PS neurons offer advantages in both accuracy and time steps. Although FS neurons can directly convert any activation function, they require pre-training of three parameters, which directly affect the conversion outcome. For the GELU activation function, FS neurons exhibit a conversion loss of 4% to 12%. On the CoLA dataset using the BERT-base model, our SNN improves accuracy by 0.38% compared to the source ANN, indicating the potential of our approach.

Table 2 Comparison of experimental results on CIFAR10, CIFAR100, and ImageNet using various methods

Method	Net. Arch.	ANN Acc.	SNN Acc.	Loss	K
ImageNet					
[17]	VGG16	71.52	70.89	−0.63	10
[16]	VGG16	73.36	72.92	−0.44	24
[9]	VGG16	74.29	72.85	−1.44	64
[39]	VGG16	74.27	73.32	−0.95	64
[11]	VGG16	73.18	72.86	−0.32	128
[40]	VGG16	69.35	65.19	−4.16	250
[37]	VGG16	73.49	73.46	−0.03	2560
[38]	VGG16	73.49	73.09	−0.40	4096
[17]	ResNet-34	73.30	73.33	0.03	10
[43]	ResNet-34	70.64	70.57	−0.07	16
[16]	ResNet-34	73.31	72.21	−1.10	24
[9]	ResNet-34	74.32	73.15	−1.17	128
[40]	ResNet-34	70.20	61.48	−8.72	250
[38]	ResNet-34	70.64	69.89	−0.75	4096
[37]	ResNet-34	70.64	69.63	−1.01	4096
[42]	EVA	89.62	89.51	−0.11	12
[17]	EfficientNetb7	85.00	83.57	−1.43	16
This work	**VGG16**	**71.52**	**71.23**	**−0.29**	**17**
This work	**ResNet-34**	**73.29**	**73.29**	**0.00**	**13**
This work	**EfficientNetb7**	**84.04**	**83.64**	**−0.40**	**13**

The header meanings are the same as in Table 1

4.3 Analysis of energy consumption

Energy efficiency is a significant characteristic and advantage of SNNs.In the energy consumption calculation process, our focus is on the energy consumption of the entire SNN during

Table 3 Comparison of experimental results on SST-2, SST-5, and CoLA using various methods

Method	Net. Arch.	ANN Acc.	SNN Acc.	Loss	K
SST-2					
[17]	RoBERTa-base	92.37	87.81	−4.56	16
[12]	RoBERTa-base	94.15	92.81	−1.34	64
This work	**RoBERTa-base**	**92.37**	**92.37**	**0.00**	**15**
SST-5					
[17]	RoBERTa-large	55.86	43.51	−12.35	16
[12]	RoBERTa-large	57.42	56.51	−0.91	128
This work	**RoBERTa-large**	**55.86**	**55.77**	**−0.09**	**17**
COLA					
This work	**BERT-base**	**81.21**	**81.59**	**0.38**	**15**

The header meanings are the same as in Table 1

Table 4 Comparison of different datasets and ANN models based on the number of spiking neurons, total spike count, timestep (K), and firing rate

Net. Arch.	Neurons(M)	Spikes(M)	K	FireRate
CIFAR10				
VGG16	0.28	0.56	16	0.18
ResNet18	0.05	0.11	16	0.21
ResNet20	0.19	0.40	16	0.19
ResNet56	0.53	2.17	16	0.32
CIFAR100				
VGG16	0.28	0.52	16	0.18
ResNet18	0.05	0.11	16	0.21
ResNet20	0.19	0.51	16	0.23
ImageNet				
VGG16	13.56	24.66	16	0.18
ResNet34	3.56	10.80	16	0.25
EfficientNetb0	6.16	24.32	16	0.31
EfficientNetb7	35.80	145.38	16	0.32

The firing rate (FireRate) is calculated as (26)

the inference process, which is one of the primary applications of SNNs. Due to differences in calculation methods, direct efficiency comparisons between different SNN models are often unfair. Like most studies, this work focuses on the spike firing rate, which is typically directly related to energy consumption. Since the parameter iteration process is performed offline before the inference begins and its computational complexity is relatively low, the energy consumption associated with this process is not considered in the experimental design. We designed experiments to calculate the spike firing rate of the PS neuron model and conducted a energy consumption analysis. We conducted experiments across datasets of varying scales and network architectures, recording key metrics such as the number of PS neurons and the total number of spikes. In our experiments, the spike rate was defined as

$$\text{FireRate} = (\frac{\text{Spikes}}{\text{Neurons}} + 1) \times \frac{1}{K} \tag{26}$$

where K represents the number of time steps used during inference. This definition reflects the spike firing rate per neuron at each time step, calculated over random input data of a specified size. After K time steps, our dynamic neuron model requires the addition of a fixed scalar b to better approximate the original activation function. This adjustment can be interpreted as each neuron ultimately emitting exactly one spike, thereby modifying the PS neuron's firing rate according to the aforementioned formula.

As shown in Table 4, the results indicate that each neuron requires only a low spike firing rate, with a maximum rate of just 0.32, to perform tasks such as image classification. Lower firing rates generally correspond to reduced energy consumption, as in neuromorphic chips, each spike is treated as an event, and only active events consume energy.

BrainScaleS-2 [44], a hybrid neuromorphic hardware combining digital and analog components, enables the use of spikes in the digital part while efficiently handling matrix multiplications in the analog part by simply disabling the leakage term in the membrane potential of the analog neu-

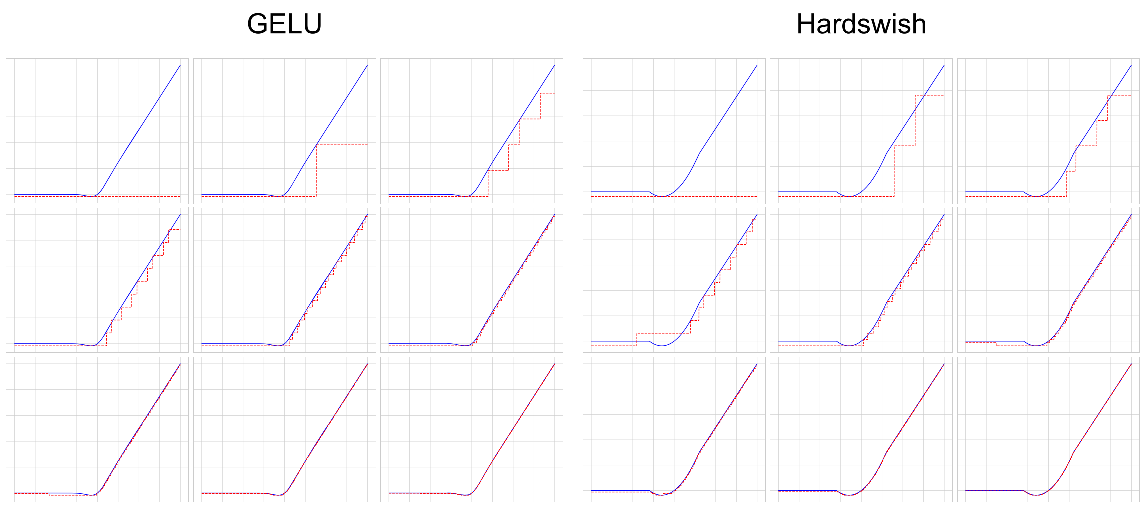

Fig. 4 Each subplot, arranged sequentially from left to right and top to bottom, illustrates the comparison between the outputs of the PS neuron and the source activation function across iterations 1 to 9. The left column corresponds to the GELU activation function, while the right column shows the HardSwish activation function. The red line represents the output values of the PS neurons in the SNN, while the blue line indicates the target output values of the corresponding activation function. These plots highlight the evolution of the PS neuron output over successive iterations, with each activation function exhibiting distinct characteristics in terms of convergence and output behavior

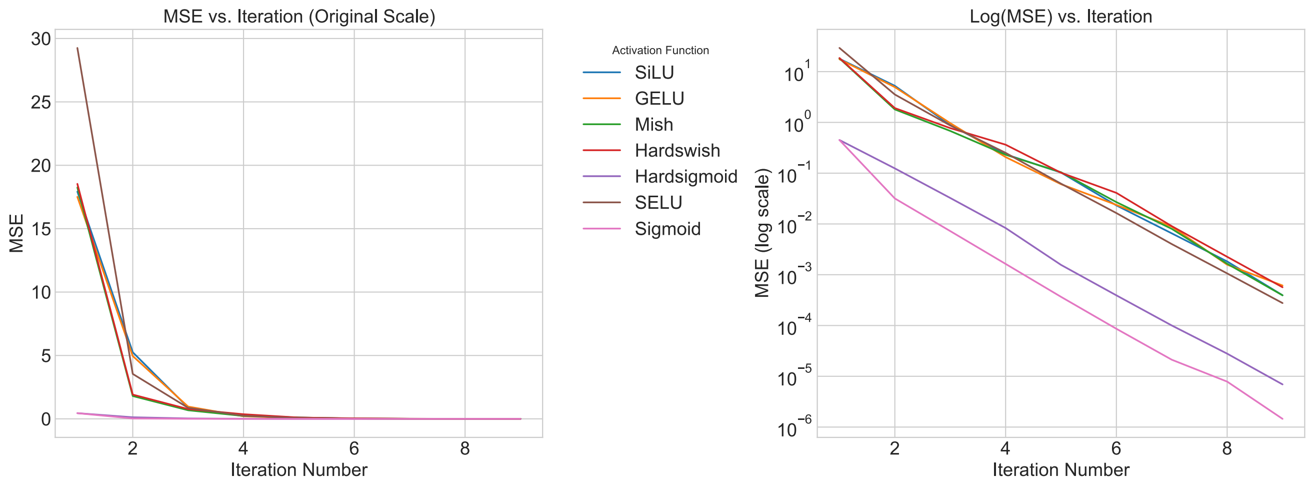

Fig. 5 Comparison of Mean Squared Error (MSE) as a function of iteration count. The left plot shows the MSE values on the original scale, while the right plot presents the logarithmic scale, illustrating an approximately linear decrease in MSE as iterations increase. This supports the notion that the fitting process converges rapidly as iterations progress

ron model. Neuromorphic chips like BrainScaleS-2 offer an optimal environment for PS neurons, as they not only capitalize on the energy-efficient nature of neuromorphic systems but also allow for rapid iteration and parameter storage of neurons. Furthermore, they support computations beyond the activation function, making them highly versatile for a broader range of model components.

4.4 Convergence analysis and generalization performance

To validate the convergence properties and generalization performance of the algorithm presented in Sect. 3.3, particularly its capability to approximate non-monotonic and piecewise functions, a series of fitting experiments were conducted. These experiments involved various activation functions and different iteration counts. The fitting results were visualized by plotting the outputs at each iteration, demonstrating how the output of the PS neuron progressively approximates the original function as the number of iterations increases. Figure 4 presents the fitting process for the GELU function (a non-monotonic function) and the HardSwish function (a piecewise function), showing the evolution of the approximation over time.

To quantitatively evaluate the fitting performance, the Mean Squared Error (MSE) between the computed output and the original target output was calculated. As shown in Fig. 5, the MSE exhibits an exponential decrease as the number of iterations increases. This behavior reflects the algorithm's ability to effectively approximate the target function over successive iterations. Table 5 presents the MSE values for each activation function at various iteration counts, confirming the consistent performance improvement with increased iterations.

In addition to the fitting experiments, further analysis was conducted using the ResNet-20 architecture on the CIFAR-10 dataset. In this experiment, the original ReLU activation function in ResNet-20 was replaced with other activation functions, and the network was trained and tested with these modifications. The converted SNNs were then evaluated,

Table 5 Mean Squared Error (MSE) values for different activation functions at various iteration counts

Activation Function	1	2	3	4	5	6	7	8	9
GELU	17.5108	4.9552	0.9677	0.2075	0.0607	0.0237	0.0088	0.0016	0.0006
Hardsigmoid	0.4500	0.1242	0.0325	0.0083	0.0016	0.0004	0.0001	0.0000	0.0000
Hardswish	18.5334	1.9127	0.7720	0.3638	0.1020	0.0409	0.0091	0.0023	0.0006
Mish	18.2491	1.8019	0.6792	0.2307	0.1044	0.0269	0.0080	0.0016	0.0004
SELU	29.2541	3.5440	0.8707	0.2514	0.0621	0.0164	0.0040	0.0011	0.0003
SiLU	17.9242	5.2522	0.8884	0.2352	0.1030	0.0232	0.0065	0.0018	0.0004
Sigmoid	0.4499	0.0318	0.0072	0.0016	0.0004	0.0001	0.0000	0.0000	0.0000

The table provides MSE values for iterations from 1 to 9, highlighting the improvement in approximation accuracy with increasing iterations

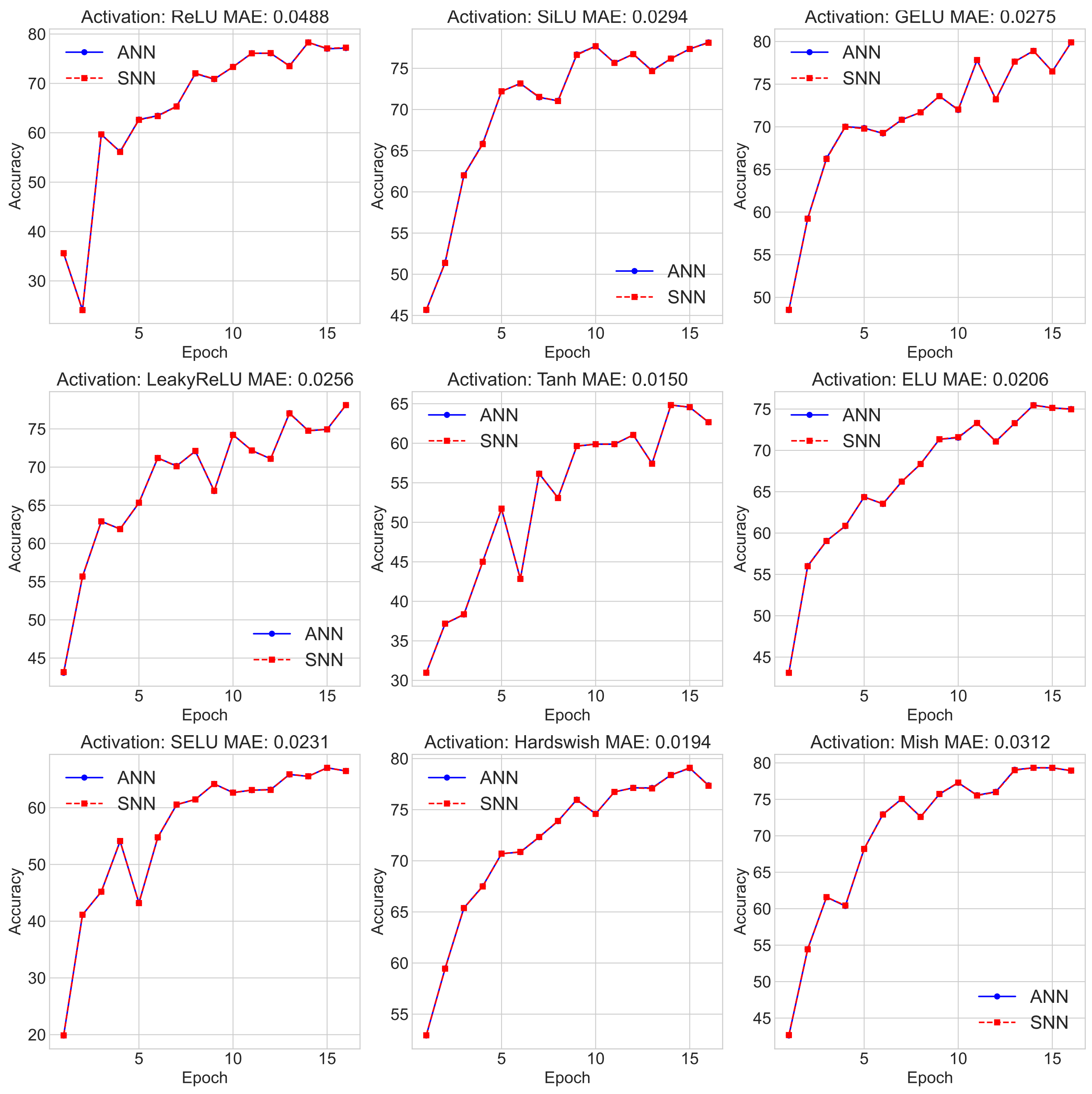

Fig. 6 Comparison of accuracy between the ANN and SNN with different activation functions at various epochs. The Mean Absolute Error (MAE) for each epoch is annotated, demonstrating that the performance of the SNN closely mirrors that of the original ANN after conversion

where the number of iterations for the PSNeuron was set to 15. The performance of the ANN and the SNN was compared across different epochs, and the Mean Absolute Error (MAE) for each epoch was computed and annotated in the corresponding figure. The results, shown in Fig. 6, indicate that the performance of the SNN is highly consistent with that of the ANN, supporting the effectiveness of using PSNeuron to replace artificial neurons in the ANN while preserving output similarity.

5 Discussion and conclusion

We introduce the Precise Spiking Neuron (PS) model, which enables high-precision and low-latency approximation of arbitrary activation functions through a streamlined iterative process. The PS model achieves state-of-the-art performance in ANN-to-SNN conversion, significantly reducing conversion loss compared to existing methods across a broad range of datasets and activation functions. In reviewing current approaches, we identified key limitations, particularly in converting non-ReLU activation functions. Our work successfully addresses these challenges, greatly expanding the applicability of ANN-to-SNN conversions by enabling accurate conversions for networks with diverse activation functions.

Furthermore, we explore the potential deployment of PS neurons on neuromorphic hardware platforms, such as BrainScaleS-2, which provide an optimal environment for the efficient execution of our model. These platforms not only support optimized parameter management but also enhance energy efficiency and computational scalability. To facilitate implementation, we offer detailed visual representations and an algorithmic workflow, making the PS neuron mechanism intuitive and straightforward to execute.

However, an important limitation arises in the process of selecting appropriate parameters for the spiking neurons during inference. This task requires the development of novel neuromorphic algorithms capable of efficiently selecting the optimal parameters from a pre-generated set, based on the specific characteristics of the input. These algorithms may include techniques such as cache storage management or spike-based search methods, which are tailored to the constraints and capabilities of neuromorphic chips. Efficient parameter selection is crucial for maintaining the performance and energy efficiency of the PS neuron model during inference. Therefore, future research should focus on the design and optimization of such algorithms to enhance the practical applicability of the PS model on neuromorphic hardware.

Acknowledgements This research is supported by National Natural Science Foundation of China (No. 62302090, 62477006, 62206046), Shanghai Sailing Program (No.23YF1401100).

Data Availability ImageNet [29], CIFAR10 [28], CIFAR100 [28], SST-2 [33], SST-5 [33], COLA [34] are publicly available datasets. No additional datasets were generated or analysed during the current study.

Code Availability The code this work is based on is publicly available https://github.com/yixi0527/PSNeuron

Declarations

Conflict of interest The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential Conflict of interest.

References

1. Li D, Chen X, Becchi M, Zong Z (2016) Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In: 2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom), IEEE, pp 477–484
2. Maass W (1997) Networks of spiking neurons: the third generation of neural network models. Neural Netw 10(9):1659–1671
3. Sun C, Sun H, Xu J, Han J, Wang X, Wang X, Chen Q, Fu Y, Li L (2022) An energy efficient stdp-based snn architecture with on-chip learning. IEEE Trans Circuits Syst I Regul Pap 69(12):5147-5158
4. Iakymchuk T, Rosado-Muñoz A, Guerrero-Martínez JF, Bataller-Mompeán M, Francés-Víllora JV (2015) Simplified spiking neural network architecture and stdp learning algorithm applied to image classification. EURASIP J Image Video Process 2015:1–11
5. Neftci EO, Mostafa H, Zenke F (2019) Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. IEEE Signal Process Mag 36(6):51–63
6. Cao Y, Chen Y, Khosla D (2015) Spiking deep convolutional neural networks for energy-efficient object recognition. Int J Comput Vision 113:54–66
7. Rueckauer B, Lungu I-A, Hu Y, Pfeiffer M, Liu S-C (2017) Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. Front Neurosci 11:682
8. Ding J, Yu Z, Tian Y, Huang T (2021) Optimal ann-snn conversion for fast and accurate inference in deep spiking neural networks. arXiv:2105.11654
9. Bu T, Fang W, Ding J, Dai P, Yu Z, Huang T (2023) Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks. arXiv:2303.04347
10. Hu Y, Zheng Q, Jiang X, Pan G (2023) Fast-snn: fast spiking neural network by converting quantized ann. IEEE Trans Pattern Anal Mach Intell
11. Wang Y, Zhang M, Chen Y, Qu H (2022) Signed neuron with memory: Towards simple, accurate and high-efficient ann-snn conversion. In: IJCAI, pp 2501–2508
12. You K, Xu Z, Nie C, Deng Z, Guo Q, Wang X, He Z (2024) Spikezip-tf: Conversion is all you need for transformer-based snn. arXiv:2406.03470
13. Wang Z, Fang Y, Cao J, Zhang Q, Wang Z, Xu R (2023) Masked spiking transformer. In: Proceedings of the IEEE/CVF international conference on computer vision, pp 1761–1771
14. Rueckauer B, Liu SC (2021) Temporal pattern coding in deep spiking neural networks. In: 2021 international joint conference on neural networks (IJCNN), IEEE, pp. 1–8
15. Kim J, Kim H, Huh S, Lee J, Choi K (2018) Deep neural networks with weighted spikes. Neurocomputing 311:373–386
16. Hwang S, Kung J (2024) One-spike snn: Single-spike phase coding with base manipulation for ann-to-snn conversion loss minimization. IEEE Trans Emerg Top Comput
17. Stöckl C, Maass W (2021) Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. Nat Mach Intell 3(3):230–238
18. Kim Y, Li Y, Park H, Venkatesha Y, Hambitzer A, Panda P (2023) Exploring temporal information dynamics in spiking neural networks. In: Proceedings of the AAAI conference on artificial intelligence, vol 37. pp 8308–8316
19. Gu Y, Gu J, Shen H, Huang K (2024) Stepwise weighted spike coding for deep spiking neural networks. arXiv:2408.17245
20. Yin B, Corradi F, Bohté SM (2023) Accurate online training of dynamical spiking neural networks through forward propagation through time. Nat Mach Intell 5(5):518–527
21. Ngu HCV, Lee KM (2022) Effective conversion of a convolutional neural network into a spiking neural network for image recognition tasks. Appl Sci 12(11):5749
22. Wu X, Zhao Y, Song Y, Jiang Y, Bai Y, Li X, Zhou Y, Yang X, Hao Q (2023) Dynamic threshold integrate and fire neuron model for low latency spiking neural networks. Neurocomputing 544:126247

23. Zambrano D, Nusselder R, Scholte HS, Bohte S (2017) Efficient computation in adaptive artificial spiking neural networks. arXiv:1710.04838
24. Nomura K, Nishi Y (2024) Synchronized stepwise control of firing and learning thresholds in a spiking randomly connected neural network toward hardware implementation. Front Neurosci 18:1402646
25. Wang C, Luo J, Wang Z (2021) A stage-wise conversion strategy for low-latency deformable spiking cnn. In: 2021 IEEE workshop on signal processing systems (SiPS), IEEE, pp 1–6
26. Zhong X, Hu S, Liu W, Huang W, Ding J, Yu Z, Huang T (2024) Towards low-latency event-based visual recognition with hybrid step-wise distillation spiking neural networks. In: Proceedings of the 32nd ACM international conference on multimedia, pp 9828–9836
27. Wang Z, Lil S, Ma Z, Yao Q (2024) High accurate, low latency conversion of spiking neural networks with blif neurons. In: 2024 IEEE 24th international conference on software quality, reliability, and security companion (QRS-C), IEEE, pp 432–440
28. Krizhevsky A, Hinton G et al (2009) Learning multiple layers of features from tiny images
29. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition, IEEE, pp 248–255
30. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
31. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556
32. Tan M, Le Q (2019) Efficientnet: Rethinking model scaling for convolutional neural networks. In: International conference on machine learning, PMLR, pp 6105–6114
33. Socher R, Perelygin A, Wu J, Chuang J, Manning CD, Ng AY, Potts C (2013) Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 conference on empirical methods in natural language processing, pp 1631–1642
34. Saeed, A., Grangier, D., Zeghidour, N.: Contrastive learning of general-purpose audio representations. In: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 3875–3879 (2021). IEEE
35. Devlin J (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805
36. Liu Y (2019) Roberta: A robustly optimized bert pretraining approach. arXiv:1907.11692
37. Han B, Roy K (2020) Deep spiking neural network: Energy efficiency through time based coding. In: European conference on computer vision, Springer, pp 388–404
38. Han B, Srinivasan G, Roy K (2020) Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 13558–13567
39. Li Y, Zeng Y (2022) Efficient and accurate conversion of spiking neural network with burst spikes. arXiv:2204.13271
40. Rathi N, Srinivasan G, Panda P, Roy K (2020) Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. arXiv:2005.01807
41. Jiang Y, Hu K, Zhang T, Gao H, Liu Y, Fang Y, Chen F (2024) Spatio-temporal approximation: A training-free snn conversion for transformers. In: The twelfth international conference on learning representations (2024)
42. Huang Z, Shi X, Hao Z, Bu T, Ding J, Yu Z, Huang T (2024) Towards high-performance spiking transformers from ann to snn conversion. In: Proceedings of the 32nd ACM international conference on multimedia, pp 10688–10697
43. Liu F, Zhao W, Chen Y, Wang Z, Jiang L (2022) Spikeconverter: An efficient conversion framework zipping the gap between artificial neural networks and spiking neural networks. In: Proceedings of the AAAI conference on artificial intelligence, vol 36. pp 1692–1701
44. Billaudelle S, Stradmann Y, Schreiber K, Cramer B, Baumbach A, Dold D, Göltz J, Kungl AF, Wunderlich TC, Hartel A, Müller E, Breitwieser O, Mauch C, Kleider M, Grübl A, Stöckel D, Pehle C, Heimbrecht A, Spilger P, Kiene G, Karasenko V, Senn W, Petrovici MA, Schemmel J, Meier K (2020) Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate. In: 2020 IEEE international symposium on circuits and systems (ISCAS), pp 1–5. https://doi.org/10.1109/ISCAS45731.2020.9180741

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Tianqi Wang undergraduate student at Donghua University, Shanghai, China. His research interests include brain-like computation, artificial intelligence and data mining.

Qianzi Shen got master degree at Tongji University, Shanghai, China. She is a researcher in China Mobile Shanghai Industry Research Institute. Her research interests include brain-like computation, artificial intelligence.

Xuhang Li is currently pursuing his M.S. degree at the School of Computer Science and Technology, Donghua University, Shanghai, China. His research focuses on brain-inspired computing architectures, spiking neural networks.

Yanting Zhang received the B.E. degree and Ph.D. degree in the School of Information and Communication Engineering from Beijing University of Posts and Telecommunications in 2015 and 2020, respectively. She used to be a visiting scholar at the University of Washington (Seattle) from 2018 to 2019. She is currently an Associate Professor in the School of Computer Science and Technology at Donghua University. Her research interests include computer vision and video/image processing.

Zijian Wang received the PhD degree from Tongji University, P.R. China. Now, he works as a lecturer in School of Computer Science and Technology, Donghua University. His research interest includes artificial intelligence and brain-inspired computing algorithms.

Cairong Yan received the Ph.D. degree in computer science and technology from Xi'an Jiaotong University, Shaanxi, China, in 2006. She is currently an associate professor and a master tutor. Her research interests include data mining, machine learning and big data processing.