Efficient Training of Spiking Neural Networks with Multi-Parallel Implicit Stream Architecture

Zhigao Cao[1*], Meng Li[1*], Xiashuang Wang[2*], Haoyu Wang[1], Fan Wang[1], Youjun Li[1], and Zi-Gang Huang[1**]

[1] Xi'an Jiaotong University, Shanxi, China
{3122113053, mengli1997, why990428, wang_fan}@stu.xjtu.edu.cn
liyoujun1@mail.xjtu.edu.cn huangzg@xjtu.edu.cn
[2] The Second Academy of China Aerospace Science and Industry Corporation, China
wxs_sky@outlook.com

Abstract. Spiking neural networks (SNNs) are a novel type of bioplausible neural network with energy efficiency. However, SNNs are non-differentiable and the training memory costs increase with the number of simulation steps. To address these challenges, this work introduces an implicit training method for SNNs inspired by equilibrium models. Our method relies on the multi-parallel implicit stream architecture (MPIS-SNNs). In the forward process, MPIS-SNNs drive multiple fused parallel implicit streams (ISs) to reach equilibrium state simultaneously. In the backward process, MPIS-SNNs solely rely on a single-time-step simulation of SNNs, avoiding the storage of a large number of activations. Extensive experiments on N-MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100 demonstrate that MPIS-SNNs exhibit excellent characteristics such as low latency, low memory cost, low firing rates, and fast convergence speed, and are competitive among latest efficient training methods for SNNs. Our code is available at an anonymized GitHub repository: https://github.com/kiritozc/MPIS-SNNs.

Keywords: Spiking Neural Networks · Memory efficiency · Energy efficiency · Low latency

1 Introduction

SNNs are considered as an important development direction of the next generation of neural networks because they have many excellent characteristics, such as low power consumption and fast response, similar to the spike coding of biological neural systems [10, 11, 20, 27]. However, SNNs convey information through spike signals, and their neuron models are non-differentiable, preventing the direct application of well-established backpropagation algorithms used in Artificial Neural Networks (ANNs) for supervised training. To address this challenge, two

* Authors contribute equally.
** Corresponding author.

main approaches have been proposed. One line of work involves converting pre-trained ANNs into SNNs [**?**, 28, 32], but this conversion process may lead to a loss of precision. A more common approach is to construct differentiable approximations of SNNs using the surrogate gradient (SG) algorithm [9, 22, 25, 30] to enable supervised training through BPTT. A consequential issue is the increase in memory costs as the simulation time steps grow during SNN training. This can result in training becoming infeasible due to hardware limitations. Additionally, the SG method accumulates errors over time, compromising model performance. However, long time step training is essential for exploring optimal model performance. Hence, it is necessary to consider a training method that addresses the memory bottleneck in SNN training and avoids the accumulation of errors caused by the SG method. Recently, the notion that stacking weight-tied neural networks converges to a fixed point has been proposed [1] and extended to the domain of SNNs [37, 38]. However, this training method requires a sufficient number of time steps for the model to converge to an equilibrium state. Additionally, the expressive capacity of the model depends on the depth of the weight-tied block. These two issues significantly constrain the training speed and inference speed of the model, posing challenges for the widespread application of SNNs in practical scenarios.

In this work, based on recent theories about equilibrium models [1, 37], we propose a novel implicit training method for SNNs. We construct an SNN composed of multiple parallel implicit streams that mutually fuse, treating the forward procedure of the network as a solver for the equilibrium state of this SNN. This solver calculates the firing rate of the model as it tends towards equilibrium, based on the model's parameters and inputs. Similar to the concept of an infinite number of layers in deep equilibrium models [1], the MPIS-SNNs in equilibrium state are equivalent to SNNs simulated for an infinite time steps. To reduce the latency of SNNs, the MPIS model reduces the depth of a single implicit stream and accelerates the convergence speed during the network's forward computation. Simultaneously, as the model converges to the equilibrium state, data gradually flows into each parallel stream, ensuring the expressive capacity of the model. In the backward process, the entire forward process is equivalent to a single iteration after reaching equilibrium. Gradient calculations are performed through implicit differentiation of the fixed-point equation, avoiding the need for storing extensive intermediate states and differentiating spikes. Additionally, we theoretically derive the rationality of employing double-bounded rectified linear unit (DBReLU) as a firing rate estimator for neurons in a single-step approximation. Our contributions include:

- We propose a training method for SNNs with constant memory costs independent of simulation time, demonstrating faster training and inference speeds, as well as lower energy consumption.
- We theoretically derive the rationale behind the DBReLU activation function as a firing rate calculation function for SNNs in equilibrium state. This activation function enables an unbiased estimation of the firing rates of neurons in equilibrium SNNs.

– We conduct extensive experiments on widely used dataset, including Fashion-MNIST, CIFAR-10, CIFAR-100, and N-MNIST. The results indicate that our method significantly saves memory compared to the BPTT method and has sparser spiking activity. In contrast to general equilibrium SNNs, our model achieves better performance with fewer time steps. Compared to the latest efficient training methods for SNNs, our approach is equally competitive.

2 Related Work

Training of Spiking Neural Networks. A classic training method with biological plausibility is the spike timing dependent plasticity (STDP) method [4,19,33,40], which adjusts synaptic weights based on the differences in spike timing between neurons, enabling self-organized feature extraction and adaptation to dynamically changing inputs without requiring global error feedback, but its performance is suboptimal. With the increasing interest in SNNs within the field of ANNs, methods have been proposed to convert pre-trained ANNs into more energy-efficient SNNs. The ANN-SNN approach leverages the correspondence between the activation functions of ANNs and the neuron models of SNNs, transforming high-performance ANNs into their corresponding SNNs [6, 11, 28, 32]. However, SNNs generated by the ANN-SNN conversion methods take a longer time to achieve high accuracy, as the consistency between ReLU activation and LIF firing rates breaks down when the time steps decrease. Another common training method involves directly training SNNs through backpropagation [9, 22, 25, 30]. To avoid non-differentiable step functions, direct training methods typically use smooth functions as substitutes for step functions during backpropagation. However, this practice introduces substitute errors, and these errors accumulate over time, affecting the model's performance. Additionally, rate-based SNNs have longer latencies compared to ANNs, and the performance of SNNs depends on this latency. And direct training requires storing intermediate activation values, which increases with the time steps, resulting in significant and potentially unacceptable memory costs. To achieve low latency and minimal memory costs, we focus on recently proposed implicit models to reduce the number of parameters and the storage of activation values [1, 7, 17], and avoid the cumulative errors caused by the SG method through the unique gradient solving approach of implicit models.

Efficient Training of Spiking Neural Networks. To address the issue of non-differentiability during the training of SNNs through backpropagation, implicit differentiation methods are often employed to circumvent non-differentiable points. [34] suggests that SNNs can be viewed as a hybrid of discrete and continuous systems, and the moments of system state transitions satisfy the local applicability of the implicit function theorem (IFT). Therefore, it becomes possible to compute jumps in their partial derivatives, enabling accurate gradient calculations and avoiding the accumulation errors caused by SG methods. Another study, through IFT, has proven the computability of gradients in SNNs and achieved

precise gradient training by leveraging causal relationships between peaks [18]. However, these approaches do not seamlessly integrate with existing sophisticated ANNs training tools and have only been validated on toy datasets. For memory and time efficiency, some training methods for SNNs typically aim to reduce model complexity or adopt approaches similar to online training. [14] reduces memory requirements by sharing leaky integrate and fire (LIF) neurons between different layers and channels. [13, 21] ignore some routes in the computational graph of SNNs as they unfold over time, reducing the time cost and memory cost of SNN training. To avoid the storage of large activation values, [39] introduces an alternative to BPTT called forward propagation through time (FPTT), which combines the dynamic spiking neuron model of liquid time constant neurons to enable online training of SNNs; [36] achieves performance comparable to offline learning by tracking presynaptic activity and utilizing instantaneous loss and gradients for forward learning. However, these methods either compromise some model performance to reduce memory consumption [6, 11, 27, 28] or require additional storage for auxiliary structures [10, 11]. Recently, some works have also adopted implicit neural networks to achieve low memory cost for SNNs [3, 5, 37, 37, 38], but their primary challenge is the longer time required to solve the implicit models, where [37] is similar to our method but does not further consider the time complexity of SNNs. Unlike other works, our approach enables training memory costs that is independent of time steps and balances the network's performance with its time complexity.

3 Preliminaries

3.1 Spiking Neural Networks

In SNNs, neurons transmit information using spikes. The spiking neurons in SNNs model the dynamics of real neural cells, receiving binary spike inputs and characterizing the received information through membrane potential and spike outputs. We typically consider the LIF model [31]:

$$\tau \frac{du(t)}{dt} = -(u(t) - u_{res}) + RI(t), u < V_{th} \tag{1}$$

where τ, $u(t)$, u_{res}, R, $I(t)$ and V_{th} denote membrane time constant , membrane potential, resting potential, membrane resistance, input current and firing threshold. When the membrane potential exceeds the threshold, the neuron fires a spike. In practical calculations, the dynamic description of neurons is represented by a discrete-time equation:

$$u_i[t] = l_i(u_i[t-1] + I[t] - V_{th}s_i[t]) \tag{2}$$

where $u_i[t]$ is the membrane potential of the neuron i after receiving input $I[t]$ at time t, l_i is the leakage factor and $s_i[t]$ is the spike output at time t. When $l_i = 1$, Eq. (2) is the integrate and fire(IF) model.

3.2 General Theory of Equilibrium Models

An important concept in neural network equilibrium is weight tying [1]. We consider an $N-$layer weight-tied model with parameters θ:

$$z^{[i+1]} = f_\theta(z^{[i]}; x), \tag{3}$$

where $z^{[i+1]}$ and x are the outputs and inputs of the current layer, respectively, $z^{[i]}$ is the hidden state of the previous layer, and f_θ is the state transition function corresponding to the weight-tied model. When the number of model layers is continuously stacked ($i.e. N \to \infty$), the relationship between input and output will satisfy the following fixed-point equation:

$$\boldsymbol{z}^* = f_\theta\left(z^*; x\right), \tag{4}$$

where z^* is the fixed point. To compute the derivative of the fixed point with respect to any parameter, we take the derivative of both sides of the Eq. (4) simultaneously, combining the chain rule of differentiation and the IFT, ultimately obtaining the target derivative:

$$\frac{\partial z^*(\cdot)}{\partial(\cdot)} = \left(J_{g_\theta}^{-1}\big|_{z^*}\right) \frac{\partial f_\theta(z^*, x)}{\partial(\cdot)}, \tag{5}$$

where $\boldsymbol{z}^*(\cdot)$ represents the implicit function of the target derivative parameters, $J_{g_\theta}^{-1}|_{z^*} = (I - \frac{\partial f_\theta(z^*,x)}{\partial z^*})^{-1}$ which can also be transformed into a fixed-point iteration problem for solving [1], and $\frac{\partial f_\theta(z^*,x)}{\partial(\cdot)}$ can be automatically computed by the deep learning framework.

4 Method

4.1 MPIS-SNNs

As is widely recognized, SNNs typically require an adequate number of time steps to ensure the accuracy of the model. SNNs with feedback connections can be approximated in the temporal dimension as a multi-layer weight-tied network with input injection (where simulation time steps correspond to the number of network layers). Therefore, we can consider SNNs as a weight-tied block and utilize the theory of the equilibrium model to regard the forward procedure of SNNs as a solver for the equilibrium state, enabling the separation of the forward and backward procedure of SNNs. In this approach, the error does not require explicit backpropagation through time, which allows for the training of SNNs with a constant memory footprint. However, a common issue present in both SNNs and equilibrium models is latency, encompassing the simulation time of SNNs and the time required to solve the fixed point in the equilibrium model. To address this concern, we employ a shallower parallel structure.

Model Architecture. The main concept of MPIS involves, on one hand, decomposing the vertical complexity of SNNs to reduce the simulation time of a

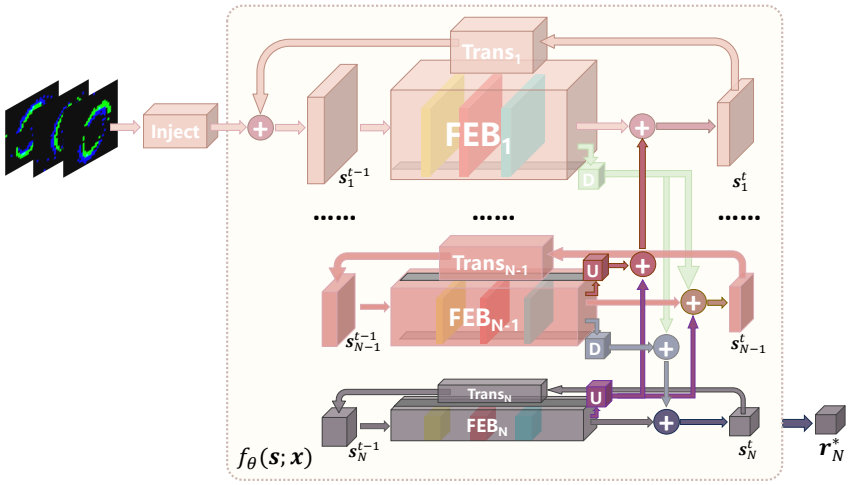


Fig. 1: MPIS Architecture. $\boldsymbol{Inject}$: Input injection module; $\boldsymbol{s_n^t}$: The output spikes of the $n-$th IS branch after t time steps of iteration; $\boldsymbol{r_N^*}$: The firing rate of the $N-$th IS after reaching the equilibrium state; $\boldsymbol{FEB_n}$: Feature extraction module of the $n-$th parallel stream branch; $\boldsymbol{Trans_n}$: Feedback module of the nth parallel stream branch; $\boldsymbol{D}$: Downsampling module of the current parallel stream; $\boldsymbol{U}$: Upsampling module of the current parallel stream.

single time step in SNNs. On the other hand, it accelerates the model's convergence by fusing each IS, thus reducing the iteration count required for solving the fixed point and achieving the goal of shortening the forward process time. The core framework of the MPIS model is illustrated in Fig. 1, where all IS collectively form the previously mentioned state transition function f_θ. The feature map sizes output by each IS gradually decrease from top to bottom. Each IS branch receives injections from other branches, and the firing rates of each IS need to reach their respective equilibrium states, *i.e.* solving for their corresponding fixed points $\boldsymbol{r}_n^*$ (where n represents the index of the IS, $n \in [1, 2, \ldots, N]$). Ultimately, multiple equilibrium states of different sizes are obtained, and the equilibrium state corresponding to the minimum-sized IS is fed into the classifier (though using equilibrium states of larger sizes for subsequent tasks is possible, it leads to increased memory cost, which is clearly undesirable). Single IS. Each IS can be viewed as an independent state transition function, consisting of a feature extraction block (FEB) and a transformation module (Trans). The input $\boldsymbol{s}_n^{t-1}$(with the initial input $\boldsymbol{s}_n^0$ being a tensor of zeros with the same shape and dimensions as paired with FEB, the reason for which will be explained below) undergoes feature extraction through FEB to obtain the output spikes $\boldsymbol{s}_n^t$ at time t. The output at time t is transformed and used as the input for time $t+1$ to undergo feature extraction through FEB once again. We use the term "*block*" instead of "*layer*" to name the state transition function because, although theoretically any single-layer network can fit the function we want to approximate [1],

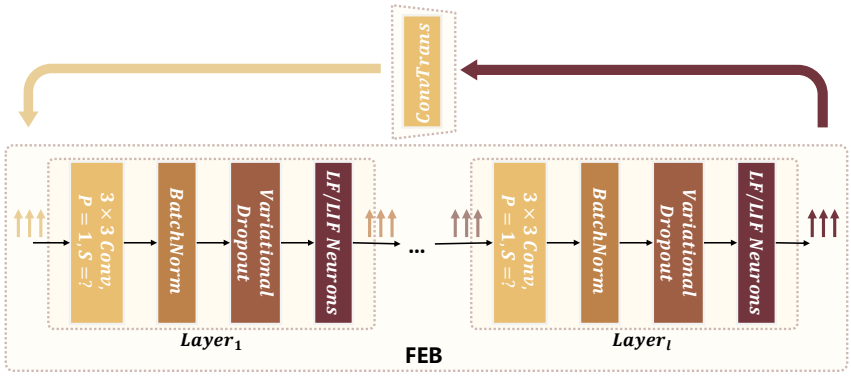


Fig. 2: Internal structure of implicit stream (IS). ***FEB***: Feature extraction block within the IS. ***Layer$_l$***: The $l-$th layer in FEB, where a sequence of similar operations is defined as one layer, including convolution, batch normalization, variational dropout, and neuron layers. ***ConvTrans***: Transformation layer that reshapes the output from the previous time step to match the shape of the input for the next time step.

a single FEB does not imply a traditional single-layer network. Instead, it is a structurally rich block composed of multiple layers, which contributes to enhancing the model's representational capacity. The network architecture is inspired by the designs in [1] and [37], and the specific design of a single IS is illustrated in Fig. 2. The FEB consists of l layers of convolutional blocks, each block sequentially containing a convolutional layer, batch normalization layer, variational dropout layer, and spiking neuron layer. However, each convolutional block has different parameter settings. The role of the Trans module is to transform the feature map output from the l-th layer of FEB back to the shape and dimensions matching the 1st layer. The Trans module provides FEB with more flexible structural configurations; otherwise, the input and output sizes of FEB must be identical.

Model Representational Capacity. While adopting our proposed shallower single IS reduces the time costs of the forward procedure, a conspicuous issue is the decrease in model depth, leading to a reduction in model complexity and, consequently, impacting the model's representational capacity, resulting in a performance decline. Therefore, if we aim to maintain sufficient performance with a reduced time costs, a natural idea is to parallelly increase the model complexity. As mentioned in Sec. 3.2, to connect the model's dynamic evolution process with the input, it is also necessary to inject the feature information of the input data into the FEB. However, we want the final equilibrium state($i.e.$ the firing rate $\boldsymbol{r}_N^*$ after the $N-$th IS reaches equilibrium) to depend on more parameters, so the input is only injected into the first IS, and the initial inputs of all ISs are set to zero. Initially, the input information only exists in the first IS, and the other ISs do not receive features about the input. The equilibrium

state of each IS depends on the injection from higher-level ISs (injected through downsampling modules), and the equilibrium states of higher-level ISs are also modulated by other ISs. The input information gradually propagates to the final IS as all ISs converge to the equilibrium point, which enables the model to have sufficient representational capacity. Additionally, to enhance the flexibility of the model, the upsampling and downsampling modules responsible for fusing the ISs can be sliding, *i.e.* they can choose to fuse at any layer of the FEB (the fusion shown in Fig. 1 occurs at the last layer of the FEB). Formally, the iteration process of the $i-$th IS can be expressed as:

$$\begin{aligned} \boldsymbol{s}_i^{t+1} = \boldsymbol{F}_i^{l+1\to L}(\boldsymbol{F}_i^{1\to l}(\boldsymbol{T}_i(\boldsymbol{s}_i^t)) + \sum_{d=1}^{i-1} \boldsymbol{D}_d(\boldsymbol{F}_d^{1\to l}(\boldsymbol{T}_d(\boldsymbol{s}_d^t))) \\ + \sum_{u=i+1}^{N} \boldsymbol{U}_u(\boldsymbol{F}_u^{1\to l}(\boldsymbol{T}_u(\boldsymbol{s}_u^t)))) \end{aligned} \tag{6}$$

where i represents the index of the IS, $i \in [1, 2, ..., N]$, l represents the number of layers in the feature extraction block, $l \in [1, 2, ..., L]$, $F_i^{a\to b}$ denotes the layers from a to b in the feature extraction block of the $i-$th IS, D_d is the downsampling module of the $d-$th IS, U_u is the upsampling module of the $u-$th IS, and l is the layer number where the fusion module is located.

4.2 Training

Forward Pass. To calculate the fixed points, one can employ acceleration methods from numerical analysis related to fixed point iterations, or convert the fixed point finding problem into a root-finding problem. However, to better align with the temporal characteristics of SNNs, here we still adopt the naive forward iteration to calculate fixed points. We denote $f_\theta(\boldsymbol{s}; \boldsymbol{x})$ as the overall state transition equation for all ISs, then the forward process can be described by an iterative equation:

$$\boldsymbol{s}^t = \lim_{t\to\infty} f_{\boldsymbol{\theta}}(\boldsymbol{s}^{t-1}; \boldsymbol{x}) \tag{7}$$

where t represents the $t-$th time step, and $\boldsymbol{s}^t$ represents the output spikes. As the simulation duration increases, the contribution of each new simulation step becomes progressively smaller until the SNNs reach equilibrium, at which point the firing rate is $\boldsymbol{r}^*$. Activation values are not retained during the forward pass. The equivalent pathway for the forward process is: $\boldsymbol{r}^*$ serves as the new input to Eq. (7) and the firing rate is calculated through single-step simulation.

Backward Pass. Let L be the loss function concerning the $\boldsymbol{r}^*$. As the neural network converges to the equilibrium state, the loss gradient with respect to the parameters $\boldsymbol{\theta}$ is

$$\frac{\partial L(\boldsymbol{r}^*)}{\partial \boldsymbol{\theta}} = \frac{\partial L(\boldsymbol{r}^*)}{\partial \boldsymbol{r}^*}(J_{g_{\boldsymbol{\theta}}}^{-1}|_{\boldsymbol{r}^*})\frac{\partial \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{r}^*, \boldsymbol{x})}{\partial \boldsymbol{\theta}}, \tag{8}$$

where $\frac{\partial L(\boldsymbol{r}^*)}{\partial \boldsymbol{r}^*}$ and $\frac{\partial \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{r}^*,\boldsymbol{x})}{\partial \boldsymbol{\theta}}$ can both be calculated along the equivalent path via automatic differentiation, and $J_{g_{\boldsymbol{\theta}}}^{-1}|_{\boldsymbol{r}^*}$ can be computed by solving a linear system [1]. The parameters are updated using the stochastic gradient descent (SGD) algorithm:

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} - \alpha \cdot \frac{\partial L(\boldsymbol{r}^*)}{\partial(\boldsymbol{\theta})}, \tag{9}$$

Where $\boldsymbol{\theta}_{old}$ and $\boldsymbol{\theta}_{new}$ are the parameters before and after the update, and α is the set learning rate. In practice, we ultimately use the firing rate $\boldsymbol{r}^T$ obtained from a finite number of simulation steps T as an approximation of $\boldsymbol{r}^*$. When the approximation error is sufficiently small, the error between the approximation and the true value has a negligible impact on the model's performance [1, 2].

4.3 Double-bounded Rectified Linear Unit

When FEB has multiple layers of neurons, the presence of step functions (spiking neurons) still prevents the direct calculation of derivatives within FEB. Inspired by the ANNs-SNNs [6, 28] and the implicit differentiation method for equilibrium SNNs [1, 37], we derive the DBReLU as the firing rate calculation function for equilibrium SNNs. We can first intuitively describe the effectiveness of our proposed method. In the research of ANNs to SNNs, the performance of the converted SNNs is directly proportional to the number of simulation time steps. However, due to hardware limitations, the simulation steps cannot be increased indefinitely. In MPIS-SNNs, the final output corresponds to the fixed point of the model, which is equivalent to the firing rates of neurons after an infinite number of time steps of simulation. IF neurons will be equivalent to unbiased estimators of linear rectification units over time. The formal expression of DBReLU is

$$r_i^l = Min\left(Max\left(0, \frac{\left(\sum_{j=1}^{M^{l-1}} W_{ij}^l r_j^{l-1}\right)}{V_{th}}\right), 1\right), \tag{10}$$

where r_i^l represents the firing rate of the $i-$th neuron in the $l-$th layer, M^{l-1} is the total number of neurons in the $(l-1)-$th layer, W_{ij}^l is the connection weight from the $j-$th neuron in the $(l-1)-$th layer to the $i-$th neuron in the $l-$th layer, and V is the threshold of the neuron. To reduce the number of inactive neurons, we have also employed techniques such as weight normalization [29] and batch normalization [12] as auxiliary measures.

Proof. Let $V(t)_i^l$ represent the membrane potential of the $i-$th neuron in the $l-$th layer at time t, then the membrane potential transformation formula for the neuron is

$$V(t)_i^l = V(t-1)_i^l + \sum_{j=1}^{M^{l-1}} W_{ij}^l s(t)_j^{l-1} - V_{th} s(t)_i^l \,, \tag{11}$$

where $s(t)_i^l$ represents the spike emitted by the $i-$th neuron in the $l-$th layer at time t, with a value of either 1 or 0. For SNNs with T steps, the following

equation is satisfied:

$$\sum_{t=1}^{T} V(t)_i^l = \sum_{t=1}^{T} V(t-1)_i^l + \sum_{t=1}^{T} \sum_{j=1}^{M^{l-1}} W_{ij}^l s(t)_j^{l-1} - V_{th} \sum_{t=1}^{T} s(t)_i^l. \tag{12}$$

The average firing rate of the $i-$th neuron in the $l-$th layer within the time period T is calculated as:

$$r_i^l = \frac{1}{T} \sum_{t=1}^{T} s\left(t\right)_i^l = \frac{1}{TV_{th}} \left(V\left(0\right)_i^l - V\left(T\right)_i^l + \sum_{t=1}^{T} \sum_{j=1}^{M^{l-1}} W_{ij}^l s\left(t\right)_j^{l-1} \right). \tag{13}$$

Let the sum of all inputs in the previous layer be denoted as $C_t = \sum_{j=1}^{M^{l-1}} W_{ij}^l s(t)_j^{l-1}$, and the membrane voltage difference at the initial and final moments as $V_\Delta = V(0)_i^l - V(T)_i^l$. When SNNs converge to a fixed point, the model is equivalent to SNNs with an infinite number of simulated time steps ($i.e. T \to \infty$). The firing rate computation is expressed as:

$$r_i^l = \lim_{T \to \infty} \frac{1}{TV_{th}} \left(V_\Delta + \sum_{t=1}^{T} C_t \right). \tag{14}$$

Since V_Δ is a constant, it follows that $\lim_{T \to \infty} \frac{1}{TV_{th}}$. The normalized floating-point values of the image data are directly used as the firing rates for the previous layer's inputs into the first layer of the MPIS-SNNs. For the neuron i in the first layer, the sum of all upstream inputs is a constant, which we denote as a_i^0. From Eq. (14), firing rate is derived that:

$$r_i^1 = \frac{a_i^0}{V_{th}}. \tag{15}$$

Following the derivation from Eq. (11) to Eq. (15) for higher layers, we obtain:

$$r_i^l = \frac{a_i^{l-1}}{V_{th}}, \tag{16}$$

where $a_i^{l-1} = \sum_{j=1}^{M^{l-1}} W_{ij}^l r_j^{l-1}$.

5 Experiments

In this section, we investigate the empirical performance of our proposed method through four parts. Firstly, we compare the performance between the BPTT training method and our approach on both a static dataset and a neuromorphic dataset, assessing metrics such as time costs, memory costs, accuracy, and firing rate (energy efficiency). Then, we compare the general equilibrium SNNs with MPIS-SNNs in terms of time efficiency and accuracy. In the third part, we discuss

the impact of multi-parallel IS on the convergence of equilibrium SNNs. Finally, we evaluate the performance disparity between MPIS-SNNs and some recently proposed efficient training methods for SNNs. Our primary aim is to demonstrate the effectiveness and practicality of MPIS-SNNs, rather than achieving state-of-the-art(sota) results in specific tasks. Nevertheless, it is evident that MPIS-SNNs exhibit significant competitiveness. The details of each experiment are provided in supplementary section 1.

5.1 Comparing with BPTT

We compare our approach with BPTT on the static dataset Fashion-MNIST [35] and the neuromorphic dataset N-MNIST [23], with the results shown in Tab. 1. It should be noted that the structures listed in the table represent the settings for a single IS, and the complete architecture comprises two such IS(the same notation is used for the architecture of MPIS-SNNs in subsequent sections). The results demonstrate that MPIS-SNNs achieve significantly higher performance with less simulation time compared to SNNs trained directly. Moreover, the memory costs of MPIS-SNNs is a constant, independent of the time steps, whereas the memory costs of BPTT increases with the simulation time and eventually becomes unacceptable. The firing rate is a critical metric for evaluating the energy consumption of SNNs. We calculate the average firing rates of trained models derived from the two training methods on the N-MNIST dataset, as illustrated in Tab. 2. The results demonstrate that MPIS-SNNs have a significantly lower firing rate compared to the BPTT training method, which corresponds to a lower energy consumption.

Table 1: Comparison of Performance on Fashion-MNIST and NMNIST.

	Method	Size	Architecture	T	Acc	Time	Memory
Fashion-MNIST	BPTT	133K	16C3-32C3	30	89.60%	31s	2.1G
			-48C3-FC10	100	89.70%	1min32s	4.8G
	MPIS	133K	16C3-32C3	**30**	**93.14%**	**30s**	**1.2G**
			-48C3-FC10	100	93.23%	1min27s	1.2G
N-MNIST	BPTT	213K	32C3-32C3	30	98.21%	1min24s	12.8G
			-64C3-FC10	100	-	-	Out of memory
	MPIS	213K	32C3-32C3	**30**	**99.31%**	**1min35s**	**3.3G**
			-64C3-FC10	100	99.27%	5min5s	3.3G

5.2 Comparing with the General Equilibrium SNNs

We construct MPIS-SNNs with the same number of parameters and a similar structure as the General equilibrium SNNs (IDE-Nets [37]) and compare their

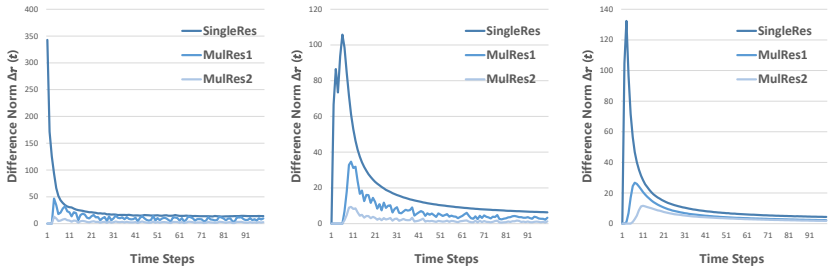
Table 2: Comparison of Firing Rate.

	Firing Rate		
	Layer1	Layer2	Layer3
BPTT	5.06e-2	7.24e-2	8.98e-2
MPIS	**8.0e-4**	**7.0e-4**	**7.3814e-5**

performance on more complex CIFAR-10 and CIFAR-100 datasets [15]. To eliminate experimental environment errors, we locally test the original code provided by [37] (for a direct comparison based on the original paper's results, refer to Section 5.4). Additionally, to explore the performance of MPIS-SNNs, we also build models with a larger number of parameters. The results are shown in Tab. 3. In terms of accuracy, MPIS-SNNs achieve higher accuracy with fewer simulation time steps. In terms of training speed, MPIS-SNNs are still faster than IDE-Net even with more parameters. Specifically, MPIS-SNNs perform better than IDE-Nets with 100 time steps on both datasets using only 30 time steps, and the time costs of MPIS-SNNs is less than half of IDE-Nets. When the parameter count exceeds IDE-Net's by twofold, MPIS-SNNs see a 2.90% increase in accuracy on CIFAR-10 and a 4.14% increase on CIFAR-100, while maintaining faster training speeds. Th highlights the potential and adaptability of MPIS-SNNs. High time efficiency and performance improvement demonstrate the potential and flexibility of MPIS-SNNs.

Table 3: Comparison of Performance on CIFAR-10 and CIFAR-100.

	Method	Size	T	Acc	Time
CIFAR-10	IDE-Nets	11.8M	30	90.37%	12min10s
			100	90.57%	22min34s
	MPIS-SNNs	11.8M	30	92.79%	2min34s
		28.5M	**30**	**93.27%**	**3min48s**
CIFAR-100	IDE-Nets	14.8M	30	70.26%	12min33s
			100	71.12%	21min50s
	MPIS-SNNs	14.8M	30	73.19%	5min33s
		30.0M	**30**	**74.40%**	**8min38s**



(a) N-MNIST **(b)** CIFAR-10 **(c)** CIFAR-100

Fig. 3: Model Convergence on Different Datasets

5.3 Convergence of MPIS-SNNs

To investigate the convergence of MPIS-SNNs, we simulate 100 time steps for MPIS-SNNs with a single IS and with two ISs, respectively, and plot the difference norm(*i.e.* $\boldsymbol{r}_{\Delta}(t) = \boldsymbol{r}(t) - \boldsymbol{r}(t-1)$) at each time. The evolution of the convergence curve is depicted in Fig. 3. The convergence curves demonstrate that MPIS-SNNs with two ISs (lighter blue) converge to equilibrium states more rapidly, and the smaller the IS branch, the quicker the convergence. This result further explains why MPIS-SNNs require fewer simulation time steps to achieve better results and utilize faster convergence to reduce the latency of SNNs.

5.4 Comparing with the Latest Efficient Training Methods

To demonstrate the effectiveness and competitiveness of MPIS-SNNs, we survey recent methods for efficient training of SNNs, which can be broadly categorized into three objectives: reducing SNNs latency [8], minimizing memory cost in SNNs training [24, 26, 36, 37, 39], and lowering energy consumption in SNNs [16, 24]. The comparative results between MPIS-SNNs and these approaches are presented in Tab. 4, with metrics derived from the optimal values reported by their respective authors. It is evident that MPIS-SNNs showcase high competitiveness across various tasks, notably achieving sota on N-MNIST, Fashion-MNIST, and CIFAR-100 datasets.

6 Conclusion

Compared to ANNs, SNNs are a type of neural network with greater biological plausibility. However, direct training of SNNs has always faced challenges such as high memory costs and non-differentiability. Additionally, the performance of SNNs depends on sufficient simulation time steps, leading to high latency. To address these issues, we propose an implicit training method for SNNs based on equilibrium model theory. During the forward procedure, we accelerate the computation speed and convergence rate of SNNs by driving multiple parallel

Table 4: Comparison of Performance on Fashion-MNIST,NMNIST,CIFAR-10 and CIFAR-100.

	Method	Architecture	T	Accuracy
N-MNIST	IDE-Net (2021) [37]	64C5-FC10	30	99.47%
	HS-IF (2023) [16]	64C3-P2-128C3-P2 -128C3-P2-FC10	15	99.44%
	MPIS	**32C3-32C3 -64C3-FC10**	**30**	**99.51%**
Fashion-MNIST	IDE-Net (2021) [37]	400-FC10	5	90.25%
	LTC-SNNs (2023) [39]	128C3-P2-128C3-P2 -FC2048-FC100-AP10	784	93.58%
	MPIS	**128C3-128C3 -256C3-FC10**	**1**	**93.83%**
CIFAR-10	IDE-Net(2021) [37]	128C3-256C3-512C3 -1024C3-FC10	100	92.82%
	Hybrid SL (2021) [16]	VGG16	100	91.29%
	Temporal Pruning (2022) [8]	VGG16	1	93.05%
	OTTT (2022) [36]	VGG11	**6**	**93.73%**
	AC2AS (2023) [32]	ResNet17	5	92.88%
	MPIS	**128C3-256C3-512C3 -1024C3-FC10**	**10**	**93.27%**
CIFAR-100	IDE-Net(2021) [37]	128C3-256C3-512C3 -1024C3-FC10	100	73.43%
	Hybrid SL (2021) [16]	VGG11	120	64.98%
	Temporal Pruning (2022) [8]	VGG16	1	70.15%
	OTTT (2022) [36]	VGG11	6	71.11%
	AC2AS (2023) [32]	ResNet17	5	73.61%
	MPIS	**128C3-256C3-512C3 -1024C3-FC10**	**5**	**74.93%**

and mutually fused shallower implicit streams to reach equilibrium simultaneously. In the backward procedure, we equivalently represent SNNs with infinite time steps in a single time step, achieving constant memory costs independent of simulation time. Furthermore, to address the non-differentiability within feature extraction block, we derive double-bounded rectified linear unit as the firing rate estimation function, avoiding the time-accumulated errors caused by surrogate gradient. Extensive experiments demonstrate the advantages of our method in terms of memory costs, network latency, accuracy, and energy consumption (spikes sparsity).

References

1. Bai, S., Kolter, J.Z., Koltun, V.: Deep equilibrium models. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Ad-

vances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019), https://proceedings.neurips.cc/paper_files/paper/2019/file/01386bd6d8e091c2ab4c7c7de644d37b-Paper.pdf
2. Bai, S., Koltun, V., Kolter, J.Z.: Multiscale deep equilibrium models. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 5238–5250. Curran Associates, Inc. (2020), https://proceedings.neurips.cc/paper_files/paper/2020/file/3812f9a59b634c2a9c574610eaba5bed-Paper.pdf
3. Bal, M., Sengupta, A.: Spikingbert: Distilling bert to train spiking language models using implicit differentiation. arXiv (08 2023)
4. Bi, G., Poo, M.: Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. JOURNAL OF NEUROSCIENCE **18**(24), 10464–10472 (DEC 15 1998). https://doi.org/10.1523/jneurosci.18-24-10464.1998
5. Bidollahkhani, M., Atasoy, F., Abdellatef, H.: Ltc-se: Expanding the potential of liquid time-constant neural networks for scalable ai and embedded systems. ArXiv **abs/2304.08691** (2023), https://api.semanticscholar.org/CorpusID:258187412
6. Cao, Y., Grossberg, S.: Stereopsis and 3d surface perception by spiking neurons in laminar cortical circuits: A method for converting neural rate models into spiking models. NEURAL NETWORKS **26**, 75–98 (FEB 2012). https://doi.org/10.1016/j.neunet.2011.10.010
7. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. p. 6572–6583. NIPS'18, Curran Associates Inc., Red Hook, NY, USA (2018)
8. Chowdhury, S.S., Rathi, N., Roy, K.: Towards ultra low latency spiking neural networks for vision and sequential tasks using temporal pruning. In: Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI. p. 709–726. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-20083-0_42, https://doi.org/10.1007/978-3-031-20083-0_42
9. Cramer, B., Billaudelle, S., Kanya, S., Leibfried, A., Grubl, A., Karasenko, V., Pehle, C., Schreiber, K., Stradmann, Y., Weis, J., Schemmel, J., Zenke, F.: Surrogate gradients for analog neuromorphic computing. PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA **119**(4) (JAN 25 2022). https://doi.org/10.1073/pnas.2109194119
10. Davies, M., Srinivasa, N., Lin, T.H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.K., Lines, A., Liu, R., Mathaikutty, D., Mccoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y.H., Wild, A., Yang, Y., Wang, H.: Loihi: A neuromorphic manycore processor with on-chip learning. IEEE MICRO **38**(1), 82–99 (JAN-FEB 2018). https://doi.org/10.1109/MM.2018.112130359
11. Deng, S., Gu, S.: Optimal conversion of conventional artificial neural networks to spiking neural networks [arxiv]. arXiv p. 14 pp. (28 Feb 2021)
12. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Bach, F., Blei, D. (eds.) INTERNATIONAL CONFERENCE ON MACHINE LEARNING, VOL 37. Proceedings of Machine Learning Research, vol. 37, pp. 448–456 (2015), 32nd International Conference on Machine Learning, Lille, FRANCE, JUL 07-09, 2015

13. Kaiser, J., Mostafa, H., Neftci, E.: Synaptic plasticity dynamics for deep continuous local learning (decolle). FRONTIERS IN NEUROSCIENCE **14** (MAY 12 2020). https://doi.org/10.3389/fnins.2020.00424
14. Kim, Y., Li, Y., Moitra, A., Yin, R., Panda, P.: Sharing leaky-integrate-and-fire neurons for memory-efficient spiking neural networks. FRONTIERS IN NEUROSCIENCE **17** (JUL 31 2023). https://doi.org/10.3389/fnins.2023.1230002
15. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Handbook of Systemic Autoimmune Diseases **1**(4) (2009)
16. Kundu, S., Datta, G., Pedram, M., Beerel, P.A.: Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression. In: 2021 IEEE WINTER CONFERENCE ON APPLICATIONS OF COMPUTER VISION WACV 2021. pp. 3952–3961. IEEE Winter Conference on Applications of Computer Vision, IEEE; IEEE Comp Soc; Adobe; Amazon; iRobot; Kitware; Verisk (2021). https://doi.org/10.1109/WACV48630.2021.00400, iEEE Winter Conference on Applications of Computer Vision (WACV), ELECTR NETWORK, JAN 05-09, 2021
17. Laborieux, A., Ernoult, M., Scellier, B., Bengio, Y., Grollier, J., Querlioz, D.: Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias. FRONTIERS IN NEUROSCIENCE **15** (FEB 18 2021). https://doi.org/10.3389/fnins.2021.633674
18. Lee, J.H., Haghighatshoar, S., Karbasi, A.: Exact gradient computation for spiking neural networks via forward propagation. In: Ruiz, F., Dy, J., van de Meent, J.W. (eds.) Proceedings of The 26th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 206, pp. 1812–1831. PMLR (25–27 Apr 2023), https://proceedings.mlr.press/v206/lee23b.html
19. Liu, F., Zhao, W., Chen, Y., Wang, Z., Yang, T., Jiang, L.: Sstdp: Supervised spike timing dependent plasticity for efficient spiking neural network training. FRONTIERS IN NEUROSCIENCE **15** (NOV 4 2021). https://doi.org/10.3389/fnins.2021.756876
20. Maass, W.: Networks of spiking neurons: The third generation of neural network models. NEURAL NETWORKS **10**(9), 1659–1671 (DEC 1997). https://doi.org/10.1016/S0893-6080(97)00011-7
21. Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., Luo, Z.Q.: Towards memory- and time-efficient backpropagation for training spiking neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 6166–6176 (October 2023)
22. 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 PROCESSING MAGAZINE **36**(6), 51–63 (NOV 2019). https://doi.org/10.1109/MSP.2019.2931595
23. Orchard, G., Jayawant, A., Cohen, G.K., Thakor, N.: Converting static image datasets to spiking neuromorphic datasets using saccades. FRONTIERS IN NEUROSCIENCE **9** (NOV 16 2015). https://doi.org/10.3389/fhins.2015.00437
24. Putra, R.V.W., Shafique, M.: Fspinn: An optimization framework for memory-efficient and energy-efficient spiking neural networks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**(11), 3601–3613 (2020). https://doi.org/10.1109/TCAD.2020.3013049
25. Qiao, G.C., Ning, N., Zuo, Y., Hu, S.G., Yu, Q., Liu, Y.: Direct training of hardware-friendly weight binarized spiking neural network with surrogate gradient learning towards spatio-temporal event-based dynamic data recognition. NEU-

ROCOMPUTING **457**, 203–213 (OCT 7 2021). https://doi.org/10.1016/j.neucom.2021.06.070
26. Qiao, G.C., Ning, N., Zuo, Y., Zhou, P.J., Sun, M.L., Hu, S.G., Yu, Q., Liu, Y.: Batch normalization-free weight-binarized snn based on hardware-saving if neuron. NEUROCOMPUTING **544** (AUG 1 2023). https://doi.org/10.1016/j.neucom.2023.126234
27. Roy, K., Jaiswal, A., Panda, P.: Towards spike-based machine intelligence with neuromorphic computing. NATURE **575**(7784), 607–617 (NOV 28 2019). https://doi.org/10.1038/s41586-019-1677-2
28. 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. FRONTIERS IN NEUROSCIENCE **11** (DEC 7 2017). https://doi.org/10.3389/fnins.2017.00682
29. Salimans, T., Kingma, D.P.: Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 29 (NIPS 2016). Advances in Neural Information Processing Systems, vol. 29 (2016), 30th Conference on Neural Information Processing Systems (NIPS), Barcelona, SPAIN, 2016
30. Siddique, A., Iqbal, M.A., Aleem, M., Islam, M.A.: A 218 gops neural network accelerator based on a novel cost-efficient surrogate gradient scheme for pattern classification. MICROPROCESSORS AND MICROSYSTEMS **99** (JUN 2023). https://doi.org/10.1016/j.micpro.2023.104831
31. STEIN, R.: Some models of neuronal variability. BIOPHYSICAL JOURNAL **7**(1), 37–& (1967). https://doi.org/10.1016/S0006-3495(67)86574-3
32. Tang, J., Lai, J.H., Xie, X., Yang, L., Zheng, W.S.: Ac2as: Activation consistency coupled ann-snn framework for fast and memory-efficient snn training. Pattern Recognition **144**, 109826 (2023). https://doi.org/https://doi.org/10.1016/j.patcog.2023.109826, https://www.sciencedirect.com/science/article/pii/S0031320323005241
33. Wu, J., Zhan, Y., Peng, Z., Ji, X., Yu, G., Zhao, R., Wang, C.: Efficient design of spiking neural network with stdp learning based on fast cordic. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I-REGULAR PAPERS **68**(6), 2522–2534 (JUN 2021). https://doi.org/10.1109/TCSI.2021.3061766
34. Wunderlich, T.C., Pehle, C.: Event-based backpropagation can compute exact gradients for spiking neural networks. SCIENTIFIC REPORTS **11**(1) (JUN 18 2021). https://doi.org/10.1038/s41598-021-91786-z
35. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017)
36. Xiao, M., Meng, Q., Zhang, Z., He, D., Lin, Z.: Online training through time for spiking neural networks. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems. vol. 35, pp. 20717–20730. Curran Associates, Inc. (2022), https://proceedings.neurips.cc/paper_files/paper/2022/file/82846e19e6d42ebfd4ace4361def29ae-Paper-Conference.pdf
37. Xiao, M., Meng, Q., Zhang, Z., Wang, Y., Lin, Z.: Training feedback spiking neural networks by implicit differentiation on the equilibrium state. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J. (eds.) ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 34 (NEURIPS 2021). Advances in Neural Information Processing Systems (2021), 35th Conference on Neural Information Processing Systems (NeurIPS), ELECTR NETWORK, DEC 06-14, 2021

38. Xiao, M., Meng, Q., Zhang, Z., Wang, Y., Lin, Z.: Spide: A purely spike-based method for training feedback spiking neural networks. NEURAL NETWORKS **161**, 9–24 (APR 2023). https://doi.org/10.1016/j.neunet.2023.01.026
39. Yin, B., Corradi, F., Bohte, S.M.M.: Accurate online training of dynamical spiking neural networks through forward propagation through time. NATURE MACHINE INTELLIGENCE **5**(5), 518+ (MAY 2023). https://doi.org/10.1038/s42256-023-00650-4
40. Zhao, Y., Lin, X., Zhang, Z., Wang, X., He, X., Yang, L.: Stdp-based adaptive graph convolutional networks for automatic sleep staging. FRONTIERS IN NEUROSCIENCE **17** (APR 20 2023). https://doi.org/10.3389/fnins.2023.1158246