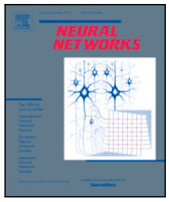
Neural Networks 172 (2024) 106121

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Full Length Article

Sampling complex topology structures for spiking neural networks

Shen Yan [a], Qingyan Meng [b,c], Mingqing Xiao [d], Yisen Wang [d,e], Zhouchen Lin [d,e,f,*]

[a] Center for Data Science, Peking University, China
[b] The Chinese University of Hong Kong, Shenzhen, China
[c] Shenzhen Research Institute of Big Data, Shenzhen 518115, China
[d] National Key Lab of General AI, School of Intelligence Science and Technology, Peking University, China
[e] Institute for Artificial Intelligence, Peking University, China
[f] Peng Cheng Laboratory, Shenzhen, 518055, China

ARTICLE INFO

Keywords:
Spiking neural networks
Neural architecture search

ABSTRACT

Spiking Neural Networks (SNNs) have been considered a potential competitor to Artificial Neural Networks (ANNs) due to their high biological plausibility and energy efficiency. However, the architecture design of SNN has not been well studied. Previous studies either use ANN architectures or directly search for SNN architectures under a highly constrained search space. In this paper, we aim to introduce much more complex connection topologies to SNNs to further exploit the potential of SNN architectures. To this end, we propose the topology-aware search space, which is the first search space that enables a more diverse and flexible design for both the spatial and temporal topology of the SNN architecture. Then, to efficiently obtain architecture from our search space, we propose the spatio-temporal topology sampling (STTS) algorithm. By leveraging the benefits of random sampling, STTS can yield powerful architecture without the need for an exhaustive search process, making it significantly more efficient than alternative search strategies. Extensive experiments on CIFAR-10, CIFAR-100, and ImageNet demonstrate the effectiveness of our method. Notably, we obtain 70.79% top-1 accuracy on ImageNet with only 4 time steps, 1.79% higher than the second best model. Our code is available under https://github.com/stiger1000/Random-Sampling-SNN.

1. Introduction

Spiking Neural Networks (SNNs), regarded as the third generation of neural networks (Maass, 1997), have attracted considerable attention due to their energy efficiency and high biological plausibility (Fang, Yu, Chen, Huang, Masquelier, & Tian, 2021; Lee, Delbruck, & Pfeiffer, 2016; Li et al., 2021; Roy, Jaiswal, & Panda, 2019; Shrestha & Orchard, 2018; Wu, Deng, Li, Zhu, & Shi, 2018; Xiao, Meng, Zhang, Wang, & Lin, 2021). The essential component of SNNs is the spiking neuron, which encodes information with binary spikes over several time steps, thus avoiding multiplication during inference. In recent years, with the development of novel training algorithms, SNNs have achieved competitive performance on several tasks, such as image classification (Meng et al., 2022), object detection (Kim, Park, Na, & Yoon, 2020), and object segmentation (Patel, Hunsberger, Batir, & Eliasmith, 2021). Meanwhile, the development of neuromorphic hardware further improves the performance of SNNs. For instance, the recently released second-generation neuromorphic research chip Loihi 2 (Orchard et al., 2021) supports larger neural architectures and new applications while providing faster and energy-efficient processing.

However, the development of SNN architectures is lagging behind. Unlike ANN, SNN has a two-dimensional computational graph containing both spatial and temporal domains. Most previous studies simply utilize ANN architectures like VGG-Net (Simonyan & Zisserman, 2015), and ResNet (He, Zhang, Ren, & Sun, 2016), ignoring the architecture gap between ANNs and SNNs. On the other hand, directly applying typical neural architecture search (NAS) methods such as ENAS (Pham, Guan, Zoph, Le, & Dean, 2018) and DARTS (Liu, Simonyan, & Yang, 2019) on searching SNN architectures can be very time-consuming due to the much slower training speed of SNNs. Existing NAS methods for SNNs (Che et al., 2022; Kim, Li, Park, Venkatesha, & Panda, 2022; Na et al., 2022) manage to accelerate the training process by imposing limitations on the search space. For instance, SNASNet (Kim et al., 2022) is selected from a cell-based search space with only four nodes in each cell. SpikeDHS (Che et al., 2022) adopts a hierarchical search space also with four nodes in each cell. AutoSNN (Na et al., 2022) searches the hyperparameters of each spiking block under a pre-defined

* Corresponding author at: National Key Lab of General AI, School of Intelligence Science and Technology, Peking University, China.
E-mail addresses: yanshen@pku.edu.cn (S. Yan), qingyanmeng@link.cuhk.edu.cn (Q. Meng), mingqing_xiao@pku.edu.cn (M. Xiao), yisen.wang@pku.edu.cn (Y. Wang), zlin@pku.edu.cn (Z. Lin).

https://doi.org/10.1016/j.neunet.2024.106121
Received 11 May 2023; Received in revised form 22 December 2023; Accepted 9 January 2024
Available online 10 January 2024

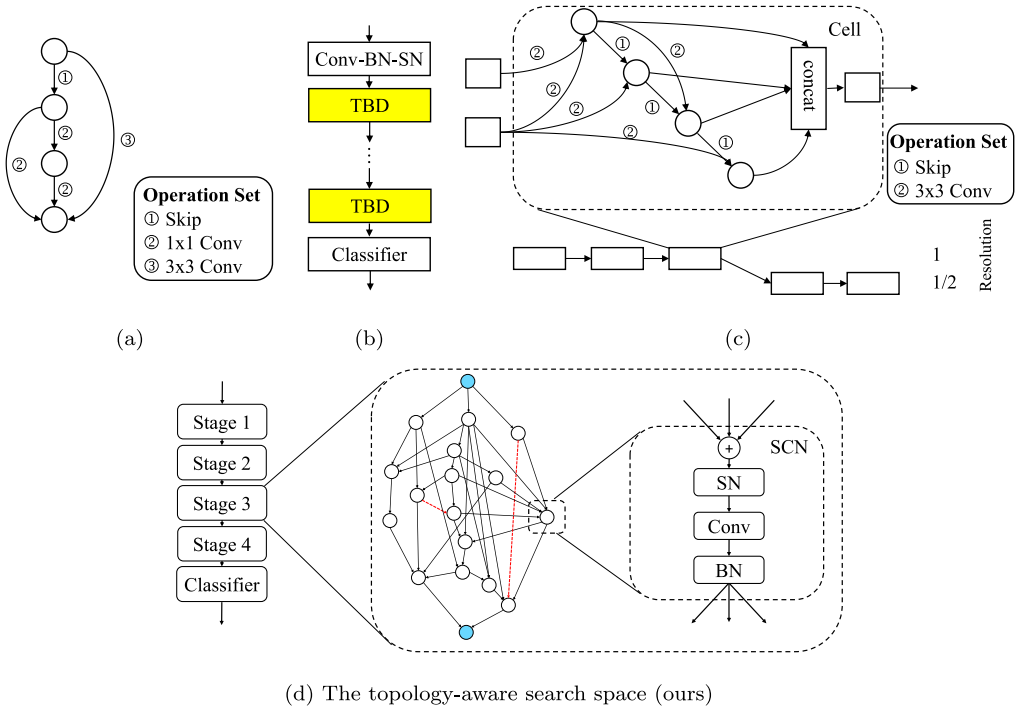


Fig. 1. A comparison of different search space designs for SNNs. (a) SNASNet (Kim et al., 2022) adopts a cell-based search space. The cell has a total of four nodes, and each edge is associated with an operation selected from the operation set. (b) AutoSNN (Na et al., 2022) employs a fixed topology while searching for the optimal choices for the to-be-determined (TBD) blocks. (c) SpikeDHS (Che et al., 2022) utilizes a search space similar to DARTS (Liu et al., 2019), with 4 nodes within a cell, and extends its search to the layer level. (d) We present the topology-aware search space enabling complex designs of spatial and temporal topologies. Our architecture comprises multiple stages, each represented by a connection topology graph. In this illustration, Stage 3 consists of 16 spiking convolution nodes (SCNs) in white, along with a source node and a sink node, both colored blue. Each SCN transforms the summation of the input data through a layer of spiking neurons (SN), a convolutional layer, and a batch normalization (BN) layer. Directed connections between nodes are indicated by arrows, while red dashed arrows represent connections with synaptic delay.

macro architecture. We note that these works do not pay enough attention to the connection topology of the SNN architecture. Their search space highly restricts the connection topology of the SNN structure, which may cause a severe loss of architecture diversity, resulting in missing the optimal SNN architecture. Therefore, it is necessary to design a dedicated search space for SNN architectures, which has a higher degree of freedom on the connection topology design.

In this study, we aim to exploit the potential of the topology design on both spatial and temporal dimensions for SNNs. To this end, we propose the topology-aware search space specifically for searching SNN architectures, which enables a more complex connection topology of the network. The topology-aware search space has a larger topology graph with at most 32 nodes, which is **eight times larger** than that in previous studies (Che et al., 2022; Kim et al., 2022). We also incorporate the synaptic delay within our architecture, thereby enabling the design of the temporal topology. This provides a novel perspective to better leverage the temporal processing ability of SNN by the architecture design. Fig. 1 illustrates the difference between our work and previous studies. Note that there is a huge number of possible connection topologies in the topology-aware search space. Learning to search for architectures in such an ample search space is a big challenge, especially due to the high training cost of SNN architectures (Kim et al., 2022; Wu et al., 2018). Nevertheless, we notice that random algorithms (Li & Talwalkar, 2019; Sciuto, Yu, Jaggi, Musat, & Salzmann, 2020; Xie, Kirillov, Girshick, & He, 2019) can also yield competitive performance compared with other NAS methods. Inspired by these works, we propose the spatio-temporal topology sampling (STTS) algorithm to obtain SNN architectures efficiently. In the proposed algorithm, we use random graph models to generate the spatial topology and sample the synaptic delay from a pre-defined distribution to generate the temporal topology. By leveraging the efficiency of random sampling, STTS avoids the search process and can obtain a powerful architecture **within 0.1 seconds**, representing a significant acceleration compared with existing NAS methods on SNNs. We evaluate our method on CIFAR-10 (Krizhevsky, Hinton, et al., 2009), CIFAR-100 (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009) datasets. Our method achieves state-of-the-art accuracy on nearly all datasets, while having a lower energy consumption compared with prior works. We summarize our contributions as follows:

1. We propose the topology-aware search space explicitly designed for SNN architecture searching. With the topology-aware search space, it is the first time that we can introduce much more diverse connection topologies into the design of SNN architectures.
2. We propose to consider the synaptic delay in the topology-aware search space, which enables a more flexible design of the temporal topology. This presents a novel perspective for exploiting the temporal learning capacity of SNNs by the architecture design.
3. We propose the spatio-temporal topology sampling (STTS) algorithm to sample architectures from our search space. Our algorithm provides an alternative way to obtain SNN architectures, avoiding the huge searching cost in previous studies.

2. Related works

There are mainly two strategies to obtain an SNN architecture: leveraging ANN architectures, and applying the NAS methods to search for SNN architectures directly.

2.1. Leveraging ANN architectures

Converting ANNs to SNNs (Kugele, Pfeil, Pfeiffer, & Chicca, 2020; Rueckauer, Lungu, Hu, Pfeiffer, & Liu, 2017) is one of the most effective methods for training SNNs. This approach leverages ANN architectures inherently. Consequently, a majority of subsequent SNN studies (Bu et al., 2022; Deng, Li, Zhang, & Gu, 2022; Han, Srinivasan, & Roy,

2020; Meng et al., 2022; Rathi, Srinivasan, Panda, & Roy, 2020; Zheng, Wu, Deng, Hu, & Li, 2021) also make use of ANN architectures. Typical ANN architectures, such as VGG-Net (Simonyan & Zisserman, 2015), and ResNet (He et al., 2016), can be adapted to SNNs by replacing the ReLU activation function with spiking neurons. Based on the ANN backbone, some SNN-friendly modifications have been proposed, such as tdBN (Zheng et al., 2021), PLIF neuron (Fang, Yu, Chen, Masquelier, Huang, & Tian, 2021), and SEW block (Fang, Yu, Chen, Huang, Masquelier, & Tian, 2021). However, naively leveraging ANN architectures is not optimal due to the inherent architectural gap between ANNs and SNNs (Kim et al., 2022).

2.2. Neural architecture search

Neural architecture search (NAS) methods are proposed for searching optimal neural architectures in an automated way. Early NAS method (Zoph & Le, 2017) defines a global search space and uses reinforcement learning (RL) as the search strategy. To search neural architectures more effectively and efficiently, recent studies in the field of NAS focus on optimization strategies, including modular search space (Zoph, Vasudevan, Shlens, & Le, 2018), continuous search strategy (Dong & Yang, 2019; Liu et al., 2019; Wu et al., 2019), weight-sharing strategy (Bender, Kindermans, Zoph, Vasudevan, & Le, 2018; Cai, Gan, Wang, Zhang, & Han, 2019; Guo et al., 2020; Na et al., 2022; Pham et al., 2018), and random algorithms (Li & Talwalkar, 2019; Sciuto et al., 2020; Xie et al., 2019). However, these methods are for searching ANN architectures only.

Recently, NAS methods have been utilized to obtain SNN architectures directly, but the related work is very limited. Kim et al. (2022) are the first to apply a NAS method for searching SNN architectures. They evaluate the representation power of each candidate architecture at initialization, thereby avoiding the training cost. Na et al. (2022) investigate design choices concerning both accuracy and number of spikes. They introduce a spike-aware evolutionary algorithm, aiming to discover an SNN architecture that achieves high accuracy and generates fewer spikes. Che et al. (2022) present a differentiable hierarchical search framework that encompasses both cell-level and layer-level search spaces. Additionally, they extend their approach to include surrogate gradient search. However, the search space in these methods is quite small, which highly restricts the topology design of the SNN architecture. In contrast, we propose to introduce a much more complex connection topology into the design of SNN architectures.

3. Preliminary

3.1. The Leaky Integrate-and-Fire (LIF) model

The fundamental component in SNNs to process binary information is the spiking neuron. Similar to previous SNN studies (Fang, Yu, Chen, Huang, Masquelier, & Tian, 2021; Kim et al., 2022; Na et al., 2022), we adopt the discrete version of the Leaky Integrate-and-Fire (LIF) model to describe the spatio-temporal dynamics of the spiking neuron, which can be formulated as

$$H^l[t] = V^l[t-1] + \frac{1}{\tau}(X^l[t] - (V^l[t-1] - V_{reset})), \tag{1}$$

$$S^l[t] = \Theta(H^l[t] - V_{th}), \tag{2}$$

$$V^l[t] = H^l[t](1 - S^l[t]) + V_{reset}S^l[t], \tag{3}$$

where l is the layer index, τ is the membrane time constant, $X[t]$ is the input current at time step t, $H[t]$ represents the membrane potential before the trigger of a spike, and $V[t]$ denotes the membrane potential after triggering. $\Theta(x)$ is the Heaviside step function. A spike is triggered if $H[t]$ exceeds the firing threshold V_{th}. We use hard reset here, which means that the spiking neuron resets its membrane potential to V_{reset} after firing a spike.

3.2. Spatio-temporal backpropagation

Due to the non-differentiability of the spike function, the training of SNN is a great challenge. Recent works on directly training SNN (Deng et al., 2022; Fang, Yu, Chen, Huang, Masquelier, & Tian, 2021) adopt the spatio-temporal backpropagation (STBP) training framework (Wu et al., 2018). These works regard the SNN as a recurrent neural network (RNN) and calculate the gradient by the backpropagation on both spatial and temporal dimensions. We use the same method, which is formulated as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial H^l[t]} \frac{\partial H^l[t]}{\partial X^l[t]} \frac{\partial X^l[t]}{\partial \mathbf{W}}, \tag{4}$$

where $\mathcal{L}$ denotes the loss function, T is the number of time steps. Since the Heaviside step function is non-differentiable, previous studies (Deng et al., 2022; Fang, Yu, Chen, Huang, Masquelier, & Tian, 2021; Wu et al., 2018) approximate its gradient with some surrogate gradients. Following these studies, we employ the same surrogate gradient as utilized in some previous works (Fang, Yu, Chen, Huang, Masquelier, & Tian, 2021; Fang, Yu, Chen, Masquelier, Huang, & Tian, 2021; Kim et al., 2022):

$$\frac{\partial S^l[t]}{\partial H^l[t]} = \frac{\alpha}{2\left[1 + \left(\frac{\pi}{2}\alpha H^l[t]\right)^2\right]}, \tag{5}$$

where α denotes the slope parameter.

4. Methodology

4.1. Topology-aware search space

The cell-based search space has been widely used in modern NAS algorithms (Dong & Yang, 2020; Liu et al., 2019; Zoph et al., 2018). In the cell-based search space, the final architecture is constructed from a few types of small cell structures, thereby significantly reducing the search complexity. However, the cell-based search space is not suitable for searching SNN architectures mainly because it highly restricts the design of the spatial topology as well as the temporal topology of the SNN architecture. Thus, designing a search space explicitly for SNN architectures is important. To this end, we propose the topology-aware search space, an SNN-friendly search space focusing on increasing the diversity of both spatial and temporal connection topologies.

As shown in Fig. 1(d), each stage in our search space can be represented by a connection topology graph. The connection topology graph consists of several spiking convolution nodes and some edges connecting the nodes. Unlike conventional representations associating operations with edges (Dong & Yang, 2020), we define operations inside each spiking convolution node, and an edge represents information transmission between a pair of nodes.

Spiking convolution node. The spiking convolution node (SCN) is the basic computing unit in the topology-aware search space. As shown in Fig. 1(d), SCN has some input edges as well as output edges. The node first receives input data from all the input edges, then aggregates all the input data via a simple summation. Afterwards, the aggregated data are transformed sequentially through a layer of spiking neurons (SN), a convolutional layer, and a batch normalization (BN) layer. Finally, the node sends out the transformed data through its output edges. We can formulate the node operations as

$$X_j[t] = \sum_{(i,j)\in E} S_i[t], \tag{6}$$

$$S_j[t] = \mathrm{BN}\left(\mathrm{Conv2d}\left(\mathrm{SN}\left(X_j[t]\right)\right)\right), \tag{7}$$

where $X_j[t]$ and $S_j[t]$ represent the aggregated data and transformed data of the jth node at time step t, respectively. In the topology-aware search space, we do not search for the choice of operations. We adopt

the SN-convolution-BN triplet inside all spiking convolution nodes. During the inference, we will merge the BN layer into the precedent convolutional layer.

Spatial connection topology. In our topology-aware search space, the spatial connection topology can be represented as a directed acyclic graph (DAG) $G = (V, E)$. For simplicity, we assume that nodes in the DAG are sorted in a topology order. An edge only points to the node with a higher index from the node with a smaller index. Benefiting from these settings, we have a high degree of freedom to design the spatial connection topology, especially when the number of nodes is large (*e.g.* $|V| = 16,\ 32$). For instance, considering a DAG with $|V|$ nodes, there are $2^{\frac{|V|(|V|-1)}{2}}$ different graphs if ignoring the isomorphism of graphs.

Temporal connection topology. Due to the spatio-temporal dynamics of the spiking neuron, SNN has not only spatial connection topology, but also temporal connection topology. The temporal connection topology of SNN, controlled by the synaptic delay, has an important impact on the effectiveness and efficiency of SNN (Maass, 1997). However, most existing works ignore the synaptic delay, restricting the utilization of temporal information. We propose to incorporate the synaptic delay into our topology-aware search space to enable temporal topology design. In our topology-aware search space, each edge is assigned an extra parameter, controlling the synaptic delay on the edge. We adopt a simplified modeling of synaptic delay in which synaptic delay is restricted to discrete values. In this model, the operations inside SCN in Eq. (6) are modified as

$$X_j[t] = \sum_{(i,j)\in E} S_i[t - d_{(i,j)}], \tag{8}$$

where $d_{(i,j)}$ is a nonnegative integer, representing the discrete synaptic delay associated with edge (i, j). The output data sent out from the ith node at time step $t - d_{(i,j)}$ arrives at the jth node at time step t through edge (i, j) after a synaptic delay of $d_{(i,j)}$. When t is smaller than $d_{(i,j)}$, we assume that $S_i[t - d_{(i,j)}]$ is a zero tensor. Additionally, $d_{(i,j)} = 0$ is possible in this model, which means that there is no synaptic delay on edge (i, j), and the operations are simplified as in Eq. (6).

Source and sink. We have defined the spiking convolution node and the connection topology graph. Besides, we need to specify the input and the output of the whole graph in order to convert the connection topology graph into a valid neural network. Note that there are still some nodes that do not have any input edge or output edge. For the nodes without any input edge, we define them as input nodes. Similarly, nodes without any output edge are defined as output nodes. We use $\mathcal{S}$ to denote the set of input nodes and $\mathcal{T}$ to represent the set of output nodes. A common approach to deal with multiple input and output nodes is to create a unique source node connecting to all the input nodes and a unique sink node receiving outputs from all the output nodes, as shown in Fig. 1(d). Specifically, we assume that the source node sends a copy of input data to every input node while the sink node collects a mean from all the output nodes. We can formulate it as

$$X_j[t] = \begin{cases} X[t], & \text{if } j \in \mathcal{S}, \\ \sum_{(i,j)\in E} S_i[t - d_{(i,j)}], & \text{otherwise}, \end{cases} \tag{9}$$

and

$$S[t] = \frac{1}{|\mathcal{T}|} \sum_{i\in\mathcal{T}} S_i[t], \tag{10}$$

where $X[t]$ and $S[t]$ denote the input and the output of the whole graph at time step t, respectively. Note that the output of the whole graph is a summation of $|\mathcal{T}|$ tensors, then be divided by $|\mathcal{T}|$. Since $|\mathcal{T}|$ is fixed as soon as the graph is determined, the division operation can be replaced by scaling up the firing threshold by $|\mathcal{T}|$ in the following spiking neurons.

Stages. It is very common (He et al., 2016; Simonyan & Zisserman, 2015; Xie et al., 2019) for neural architectures to have multiple stages to down-sample the feature maps from high resolution to low resolution, especially for image classification and object detection. Inspired by these works, We divide our entire model into several stages that generate different sizes of feature maps. Each stage is defined by a connection topology graph described above. For two adjacent stages, the preceding stage's sink node is also the succeeding stage's source node.

Regarding image input, we assume that the input resolution for the ith stage is $H_i \times W_i$ with C_i channels. To make a 2× downsampling of the resolution, we use a stride of 2 and an output channel number of $2C_i$ in the convolutional layer inside each input node. For other nodes in the same stage, a stride of 1 is used, and the input and the output channel numbers are both $2C_i$. By doing so, the output of the ith stage has a dimension of $2C_i$ with a resolution of $\frac{H_i}{2} \times \frac{W_i}{2}$.

The class of architectures defined by the topology-aware search space is named the topology-aware network (TANet). We adopt a similar macro skeleton as previous studies (Xie et al., 2019). Table 1 summarizes two settings of our TANet: the tiny version, referred to as TANet-Tiny, and the regular version, referred to as TANet-Regular. The TANet-Tiny has four stages in total. It is designed for datasets with a smaller resolution, such as CIFAR-10 and CIFAR-100. The TANet-Regular has five stages and is proposed for larger resolution datasets such as ImageNet. Both TANet-Tiny and TANet-Regular have a classifier in the end.

Table 1
Two settings of TANet. Each stage is generated from a connection topology graph. For each graph, N denotes the number of nodes, and C represents the output channel. We set $N = 1$ for the first few stages, where the graph becomes a single node.

Stage	TANet-Tiny	TANet-Regular
1	single node C	single node $C/2$
2	single node C	graph $N/2,\ C$
3	graph $N,\ 2C$	graph $N,\ 2C$
4	graph $N,\ 4C$	graph $N,\ 4C$
5	–	graph $N,\ 8C$
	classifier	

4.2. Spatio-temporal topology sampling

Modern NAS methods (Liu et al., 2019; Pham et al., 2018) have already achieved state-of-the-art performance on many tasks. However, these methods are not suitable for searching SNN architectures from the topology-aware search space, due to the huge number of candidate architectures in the search space and the high training cost of directly training SNNs (Kim et al., 2022). Nevertheless, recent works on random sampling (Sciuto et al., 2020; Xie et al., 2019) have given us an alternative approach to yield powerful neural architectures without high searching costs. In these works, neural architectures are sampled from the search space through some pre-defined random algorithms. Inspired by these studies, we propose the spatio-temporal topology sampling (STTS) algorithm to sample SNN architectures from the topology-aware search space randomly. Different from random search algorithm (Li & Talwalkar, 2019), STTS does not evaluate the performance of candidate architectures but directly outputs the sampled architecture as the final architecture. Therefore, STTS is very efficient compared with other NAS methods.

Specifically, in STTS, we sample the spatial topology as well as the temporal topology for each stage. As shown in Algorithm 1, STTS consists of two procedures: (1) using random graph models to generate

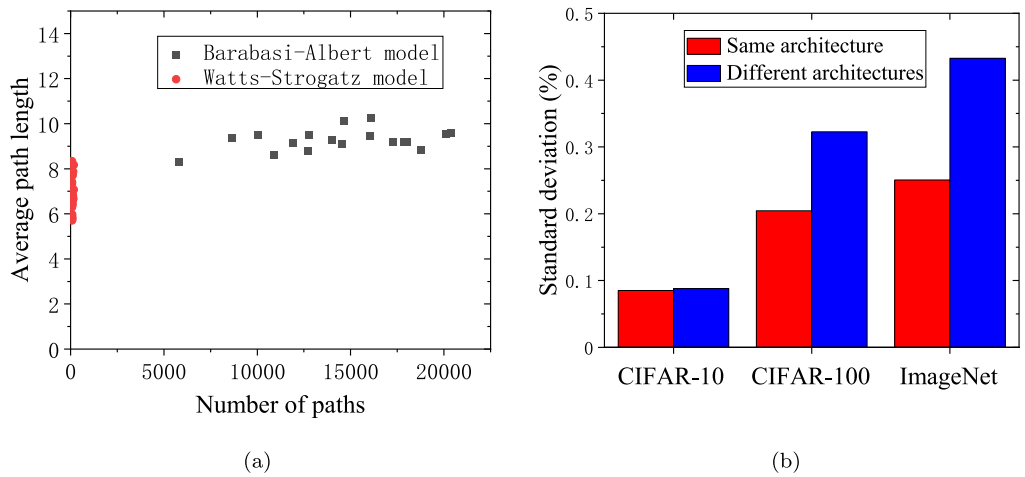


Fig. 2. Similarity between different samples. (a) The number of paths and the average path length across different samples. (b) The standard deviation of the accuracies across several training runs of the same architecture and across different samples.

the spatial topology and (2) sampling the synaptic delay of each edge from a pre-defined distribution to generate the temporal topology. As soon as we determine the connection topology graph, we convert the graph into a valid neural network. Finally, we stack stages sequentially and output the final architecture.

In practice, we use the Watts–Strogatz (WS) model (Watts & Strogatz, 1998) and the Barabási–Albert (BA) model (Albert & Barabási, 2002) to generate the spatial topology in the first procedure. We describe the details in Appendix A. Since the random graph models generate undirected graphs, we have to convert the undirected graph to a DAG. To do so, we randomly assign a topology order. Then each edge is directed from the node with a smaller index to the node with a higher index. In the second procedure, we randomly sample the synaptic delay of each edge independently from a pre-defined distribution D_{sd}. We set the synaptic delay to be either 0 or 1, and D_{sd} is a Bernoulli distribution. The parameter p in the Bernoulli distribution is referred to as the synaptic delay parameter, which controls the temporal connection topology.

Although STTS directly outputs the final architecture without evaluation, as we will see in the experimental results section, sampling multiple architectures leads to similar accuracies after training.

Algorithm 1: Spatio-temporal topology sampling

Input: Random graph model; Number of stages n; Synaptic delay distribution D_{sd};
for $i = 1$ *to* n **do**
  Generate spatial topology $G_i = (V_i, E_i)$ through the random graph model;
  foreach e *in* E_i **do**
    Sample the synaptic delay of edge e from D_{sd};
  end
  Covert the connection topology graph into a valid neural network;
end
Stack each stage sequentially to construct the final architecture.

5. Experiments

We conduct extensive experiments to verify the effectiveness of our proposed method. The implementation details of these experiments are described in Appendix B.

5.1. Architecture details

We use two settings of TANet in our experiments: TANet-Tiny for CIFAR-10/100 and TANet-Regular for ImageNet. We set the number of nodes N to 32 and the number of channels C to 96 for both TANet-Tiny and TANet-Regular. For the synaptic delay parameter p, we set it to 0.05 for both TANet-Tiny and TANet-Regular. We use the BA graph model and the WS graph model to generate the spatial topology of TANet-Tiny and TANet-Regular, respectively. Between the final stage and the classifier, there is a layer of spiking neurons. The classifier consists of a 1×1 convolution-BN-SN triplet, a global average pooling layer, and a voting layer. To avoid overfitting, We add a dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) layer between the average pooling layer and the voting layer in the classifier of TANet-Tiny, and we set the drop probability to 0.2.

We use depthwise separable convolution (Chollet, 2017) in every spiking convolution node. The depthwise separable convolution consists of two layers: the depthwise convolution and the pointwise convolution. The depthwise convolution has a kernel size of 3×3 while the pointwise convolution has a kernel size of 1×1. Note that we can transform the depthwise separable convolution into a standard convolution with the same function. Therefore, we use depthwise separable convolutions in the training process and replace them with equivalent convolutions during the inference.

5.2. Similarity between different samples

In the STTS algorithm, we sample architecture once without evaluation. This raises the question of whether different samples will exhibit significant variation in accuracy after the training process. To address this concern, we empirically demonstrate the similarities between different samples and show that sampling multiple architectures leads to similar accuracies after training.

It is noteworthy that the architectures we sampled with different seeds all utilize the same graph model for generating spatial topology and the same distribution for generating temporal topology. Consequently, these samples inherently exhibit similarities. While previous research (Albert & Barabási, 2002) has provided theoretical analyses of the properties of random graph models, we empirically quantify these inherent similarities among our samples. Specifically, we calculate two important statistical values of the spatial topology: the number of paths and the average path length. These values have been shown to exhibit a strong correlation with architecture performance in previous studies (White, Neiswanger, & Savani, 2021; You, Leskovec, He, & Xie,

Table 2
Comparison between our work and other methods on CIFAR-10, CIFAR-100, and ImageNet. "Params" denotes the number of parameters in the architecture. We categorize methods into two groups: training methods and architectural designs. For each dataset, the first section presents methods related to training techniques, while the second section focuses on architectural designs. Accuracy is reported as mean and standard deviation (after ±) based on five runs with different random seeds.

	Method	Architecture	Params	Time steps	Accuracy (%)
CIFAR-10	Rathi et al. (2020)	VGG-9	32M	100	90.54
	Yan, Zhou, and Wong (2021)	VGG-like	9M	600	94.16
	Zheng et al. (2021)	ResNet-19	13M	6	93.16
	Deng et al. (2022)	ResNet-19	13M	4	94.44
	Kim et al. (2022)	Searched Architecture	20M	5	92.73
	Na et al. (2022)	Searched Architecture	21M	8	93.15
	Che et al. (2022)	Searched Architecture	14M	6	**95.50**
	STTS (ours)	TANet-Tiny	7M	4	95.10 ± 0.09
CIFAR-100	Rathi et al. (2020)	VGG-11	37M	125	67.87
	Yan et al. (2021)	VGG-like	9M	300	71.84
	Deng et al. (2022)	ResNet-19	13M	4	74.47
	Kim et al. (2022)	Searched Architecture	21M	5	73.04
	Na et al. (2022)	Searched Architecture	5M	8	69.16
	Che et al. (2022)	Searched Architecture	14M	6	76.25
	STTS (ours)	TANet-Tiny	7M	4	**76.33 ± 0.32**
ImageNet	Rathi et al. (2020)	VGG-16	138M	250	65.19
	Rathi and Roy (2021)	VGG-16	138M	5	69.00
	Deng et al. (2022)	SEW-ResNet-34	22M	4	68.00
	Fang, Yu, Chen, Huang, Masquelier, and Tian (2021)	SEW-ResNet-50	26M	4	67.78
	Che et al. (2022)	Searched Architecture	58M	6	68.64
	STTS (ours)	TANet-Regular	25M	4	**70.79 ± 0.43**

2020). The results are presented in Fig. 2(a). Notably, instances sampled from the same graph model exhibit clear similarities. Specifically, instances sampled from the WS model demonstrate similar numbers of paths, while those sampled from the BA model exhibit comparable average path lengths.

Additionally, we calculate the standard deviation across multiple training runs of the same architecture and compare it to the standard deviation across different samples. The results are shown in Fig. 2(b). Notably, the variation across different samples is slightly larger than that observed across several training runs of the same architecture. However, it is important to highlight that in both cases, the standard deviation remains relatively low.

5.3. Comparison to the state of the art

We compare our experimental results with some SOTA methods on CIFAR-10, CIFAR-100, and ImageNet. The results are summarized in Table 2.

For the CIFAR-10 dataset, our architecture achieves competitive accuracy among all other methods. The reported mean accuracy of TANet-Tiny is only 0.4% lower than the SpikeDHS (Che et al., 2022), although TANet-Tiny uses a much less number of parameters and time steps. The results also show that the variation of the corresponding accuracies among the sampled architectures is low. The classification accuracy only has a standard deviation of 0.09% among these samples.

We can see that our method has a significant improvement on more difficult datasets. For the CIFAR-100 dataset, our architecture yields state-of-the-art results using only 4 time steps. Our method outperforms the works on training methods (first section of the results on each dataset) and those on architectural designs (second section of the results on each dataset). Specifically, TANet-Tiny achieves a mean accuracy of 76.33%, which performs 0.08% advance compared to SpikeDHS.

For the ImageNet dataset, our architecture has 70.79% top-1 accuracy using 4 time steps, achieving a 3.01% increment compared with SEW-ResNet-50 and a 2.15% increment compared with SpikeDHS. Note that our architecture has fewer parameters and uses a similar or much less number of time steps.

Table 3
Comparison with SEW-ResNet backbone on ImageNet. "w/o SN" and "w/ SN" represent the SNN architecture and the ANN architecture using the same backbone, respectively. Accuracy is reported as mean and standard deviation (after ±) based on five runs with different random seeds.

Architecture	w/o SN (%)	w/ SN (%)
SEW-ResNet-50	$76.34_{\pm 0.12}$	$66.16_{\pm 0.11}$
TANet-Regular (ours)	$74.93_{\pm 0.27}$	$70.79_{\pm 0.43}$

5.4. Analysis on the topology-aware search space

We conduct experiments to validate the suitability of our topology-aware search space for obtaining SNN architectures. In our comparison, we evaluate TANet-Regular alongside SEW-ResNet-50 (Fang, Yu, Chen, Huang, Masquelier, & Tian, 2021), which is an SNN-friendly modification of the conventional ANN architecture ResNet (He et al., 2016). To ensure a fair comparison, we use identical hyperparameters and training methods for both models. The results are shown in Table 3. Under the same training settings, TANet-Regular achieves a 4.63% higher classification accuracy compared to SEW-ResNet-50 in the context of SNNs. These experimental results strongly demonstrate that typical ANN architectures are not optimal for SNNs. Our topology-aware search space proves to be an SNN-friendly search space that can further exploit the potential of SNN structures.

5.5. Effect of the size of the graph

We test the effect of the size of the graph in the topology-aware search space. In details, We vary the number of nodes in the graph from 8 to 64 while adjusting the number of channels to keep the total number of parameters. The results are shown in Table 4. We can see that as we increase the number of nodes from 8 to 32, the performance improves, implying that SNN architectures benefit from a more complex connection topology. However, as the number of nodes continuing increasing from 32 to 64, the accuracy becomes lower and lower. This might be because the decreasing of the number of channels has a more significant impact as we increase the number of nodes. Additionally, increasing the number of nodes above a certain level might lead to overfitting.

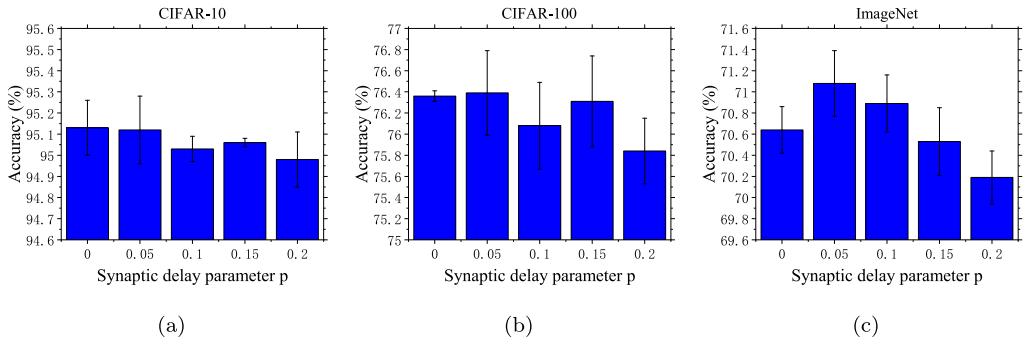


Fig. 3. Effects of the synaptic delay parameter p on (a) CIFAR-10, (b) CIFAR-100, and (c) ImageNet.

Table 4
Effect of the size of the graph on CIFAR-10. Accuracy is reported as mean and standard deviation (after ±) based on three runs with different random seeds.

Number of nodes	8	16	32	48	64
Accuracy (%)	$94.81_{\pm 0.07}$	$94.93_{\pm 0.07}$	$94.99_{\pm 0.04}$	$94.58_{\pm 0.12}$	$94.27_{\pm 0.11}$

5.6. Effect of the synaptic delay

We conduct experiments to analyze the influence of the synaptic delay parameter p. We vary p from 0 to 0.2, where $p = 0$ means no synaptic delay in the architecture. For values of p greater than 0.2, we observe that it is not suitable for the static image classification task, since the number of time steps is small (*e.g.* $T = 4$). The results are shown in Fig. 3. We can see that the architecture is not optimal when there is no synaptic delay. When we increase p to 0.05, we observe a slight improvement in classification accuracies for CIFAR-100 and ImageNet. However, as we continue increasing the synaptic delay parameter p (*e.g.* $p = 0.15, 0.2$), an increasing number of spikes are truncated, leading to a noticeable decrease in accuracy. We choose $p = 0.05$ for optimal on all the datasets.

5.7. Energy efficiency

In contrast to ANNs, SNNs perform event-based operation and multiplication-free inference. In this section, we demonstrate the energy efficiency of our method by measuring the sparsity and energy consumption of TANet.

Sparsity. To provide a comprehensive insight into the sparsity of our architecture, we calculate both the average firing rate (per time step) and the spike count for each SCN in our architecture. Fig. 4 visualizes the sparsity statistics at different stages. The experimental results reveal a notably sparse spiking activity within our architecture. Specifically, concerning firing rates, the majority of nodes exhibit average firing rates of less than 10% in TANet-Tiny and less than 15% in TANet-Regular. In terms of spike counts, the majority of nodes generate fewer than 30 K spikes in TANet-Tiny and 500 K spikes in TANet-Regular. Both of these two sparsity statistics exhibit distinct patterns at different stages, with the firing rate and spike count decreasing as the neural network's depth increases.

Energy consumption. Following previous studies (Che et al., 2022; Deng et al., 2022; Lemaire et al., 2022; Li et al., 2021), we estimate the energy consumption of our architecture by measuring the operational cost and the memory cost. For more details of the energy consumption analysis, please refer to Appendix C. We conduct a comprehensive analysis of energy consumption, comparing our architectures with typical ANN architecture, SNN architecture searched by SpikeDHS (Che et al., 2022), and ANN versions of our architectures. The results are shown in Table 5 and Table C.7. We can see that our architectures achieve the lowest energy consumption among these architectures.

Table 5
The operational cost. "# Add." and "#Mult." represent the number of addition and multiplication operations, respectively. An MAC operation leads to an increase in both #Add. and #Mult., while an AC operation only increases #Add.

	Architecture	#Add.	#Mult.	Operational cost (mJ)
CIFAR-10	ResNet-19 (ANN)	2285M	2285M	10.5
	TANet-Tiny (ANN)	735M	735M	3.4
	SpikeDHS (SNN)	5973M	19M	5.5
	TANet-Tiny (ours)	1329M	3M	**1.2**
ImageNet	ResNet-50 (ANN)	4134M	4134M	19.0
	TANet-Regular (ANN)	3093M	3093M	14.2
	TANet-Regular (ours)	9026M	16M	**8.2**

6. Conclusion and future work

In this paper, we propose the topology-aware search space to expand the range of design choices for SNN architectures. The topology-aware search space not only enables complex designs of spatial topology but also empowers the design of temporal topology by incorporating the synaptic delay. Instead of utilizing a classical NAS method, we propose the spatio-temporal topology sampling (STTS) algorithm to obtain SNN architectures from our search space. By avoiding the unaffordable NAS cost, our algorithm is much more efficient compared with previous studies. Experimental results show that our proposed method achieves state-of-the-art accuracy and generates fewer spikes on image classification tasks. Our method highlights the significance of the connection topology in designing SNN architectures.

For our work, there is still a lack of theoretical guarantee of the random sampling method, although empirical results show that the variance between different sampled architectures is low. Additionally, we only investigate the benefits of our approach on the static image classification tasks. These tasks do not adequately demonstrate the benefits of random delays. In future research, we plan to analyze the effectiveness of different topologies in SNNs theoretically. We will also investigate the correlation between the spatial topology and the temporal topology on more diverse tasks or datasets. We aim to propose a robust NAS method with a theoretical guarantee and yield more powerful SNN architectures.

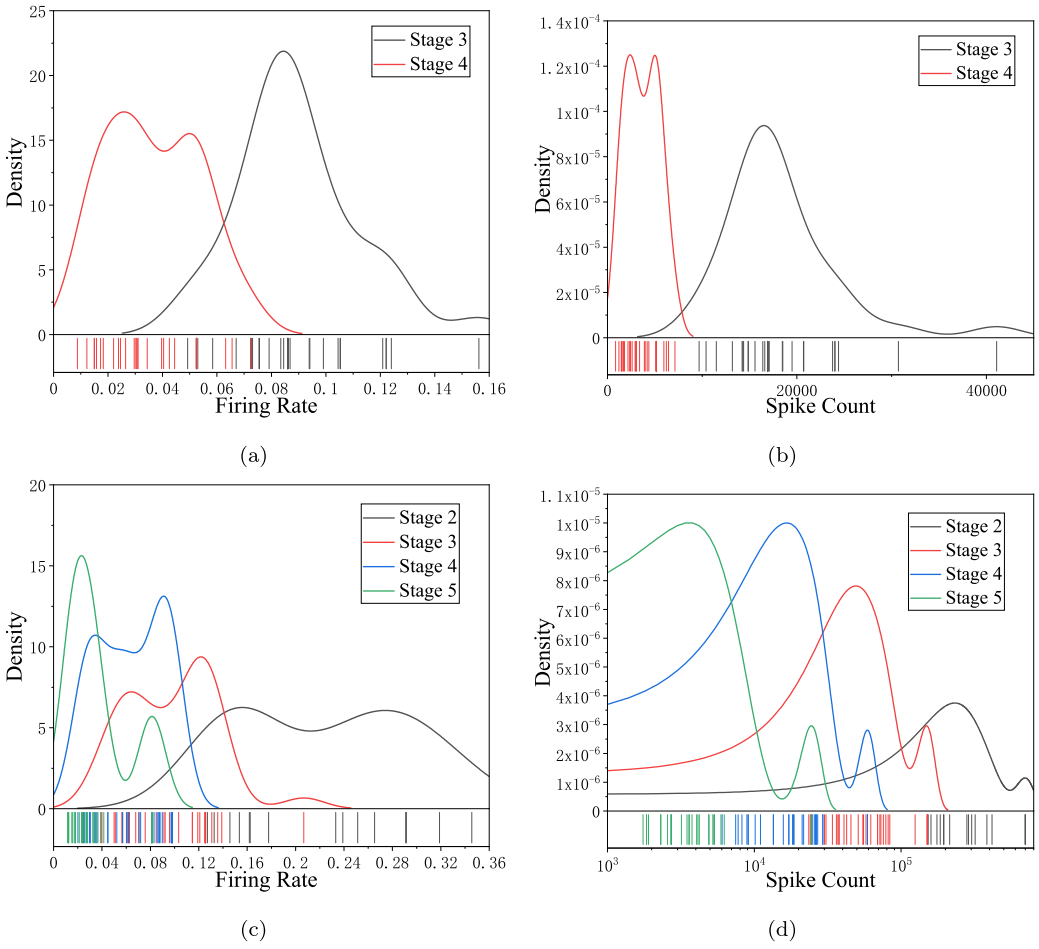


Fig. 4. Sparsity statistics at different stages. (a) Firing rate distribution in TANet-Tiny. (b) Spike count distribution in TANet-Tiny. (c) Firing rate distribution in TANet-Regular. (d) Spike count distribution in TANet-Regular.

CRediT authorship contribution statement

Shen Yan: Conceptualization, Investigation, Methodology, Writing – original draft, Writing – review & editing. **Qingyan Meng:** Conceptualization, Writing – review & editing. **Mingqing Xiao:** Writing – review & editing. **Yisen Wang:** Writing – review & editing. **Zhouchen Lin:** Funding acquisition, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

Z. Lin was supported by National Key R&D Program of China (2022ZD0160302), the NSF China (No. 62276004), and the major key project of PCL, China (No. PCL2021A12).

Appendix A. Random graph models

To generate network topologies, Xie et al. (2019) use random graph models from graph theory. Inspired by their works, we use the Watts–Strogatz (WS) model (Watts & Strogatz, 1998) and the Barabási–Albert (BA) model (Albert & Barabási, 2002) to generate the spatial topology. We describe them in the following.

Watts–Strogatz. The WS model is a random graph model which can generate graphs with small-world properties. To generate an undirected graph with N nodes, the WS model contains two procedures: (1) Place the N nodes in a regular ring lattice. Each node is connected to K neighbors, with $K/2$ on both sides. (2) For every node v, the edge that connects v to its $K/2$ rightmost neighbors is rewired with probability P. "Rewire" means that the edge is replaced with edge (v, k) where the node k is uniformly chosen at random while avoiding duplication.

Barabási–Albert. The BA model is a random graph model which generates scale-free networks. The graph begins with M nodes without any edges. Then, new nodes are sequentially added to the graph, one at a time. Each new node will be connected to M existing nodes with a probability proportional to the degrees of the existing nodes.

In our experiments, we set $K = 4, P = 0.75$ for the WS model and $M = 8$ for the BA model. Consequently, in the case of TANet-Tiny, a graph with 32 nodes exhibits dense connectivity, with a total of 192 edges. In the configuration of TANet-Regular, a 32-node graph has 64 edges, representing a sparser connectivity pattern. We visualize two examples in Fig. A.5.

The choice of the graph model for each network is based on our experimental results. In the case of TANet-Tiny, we observe a slight

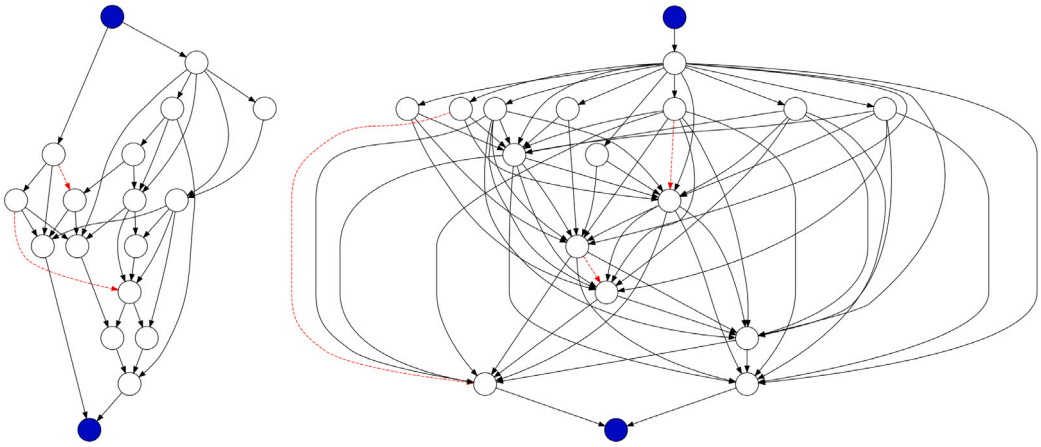


Fig. A.5. Two topology graph instances. (a) A topology graph with the spatial topology generated by the WS model. (b) A topology graph with the spatial topology generated by the BA model. Nodes representing the source and sink are highlighted in blue, while red dashed arrows represent connections with synaptic delays.

Table B.6
Hyperparameters for training on CIFAR-10/100 and ImageNet.

Dataset	CIFAR-10	CIFAR-100	ImageNet
Optimizer	SGD	SGD	SGD
Epoch	300	300	120
lr	0.1	0.1	0.1
lr scheduler	Cosine Annealing	Cosine Annealing	Cosine Annealing
Weight decay	1e−4	1e−4	4e−5
Batch size per GPU	64	64	16
GPU	1	1	16

improvement in accuracy (about 0.1%) when applying the BA model compared to the WS model. Therefore, we use the BA model for TANet-Tiny. For TANet-Regular, the BA model and the WS model perform equally well. As a result, we select the WS model for TANet-Regular, as it generates a simpler graph structure.

Appendix B. Implementation details

B.1. Datasets

We conduct experiments on CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009).

CIFAR. The CIFAR dataset consists of 60k colored images with a resolution of 32 × 32. The images are separated into 50k training samples and 10k test samples. There are 10 classes of objects in CIFAR-10 and 100 classes of objects in CIFAR-100. We apply the same data preprocessing for CIFAR-10 and CIFAR-100, which contains data normalization, random horizontal flipping, cutout (DeVries & Taylor, 2017), and random cropping. We employ direct encoding (Rathi & Roy, 2021) to convert image pixels into time series. In this method, the floating-point pixel matrix of the images is duplicated at each time step and then fed into the first layer of the SNN architecture.

ImageNet. The ImageNet dataset consists of over 1250k training images, 50k validation images, and 100k test images. We apply data normalization so that the input data have zero mean and unit variance. Besides, our data preprocessing contains random resized cropping and horizontal flipping. The input size is set to 224 × 224 by default. We also use the direct encoding (Rathi & Roy, 2021), as done for the CIFAR dataset.

B.2. Training settings

Our implementation is based on PyTorch (Paszke et al., 2019). We use SpikingJelly (Fang et al., 2020) to implement the LIF neuron. Specifically, we consistently set the number of time steps to 4 throughout our experiments. We set $\lambda = 3.0$, $V_{reset} = 0$, and $V_{th} = 1$ in every spiking neuron for all datasets. The surrogate gradient used in this work can be formulated as

$$\frac{\partial S^l[t]}{\partial H^l[t]} = \frac{\alpha}{2\left[1 + \left(\frac{\pi}{2}\alpha H^l[t]\right)^2\right]}, \tag{B.1}$$

where we set $\alpha = 2$ for all datasets. We use the cross-entropy loss and adopt the standard STBP (Wu et al., 2018) training framework to train the SNN architecture. As recommended by Zenke and Vogels (2021), We detach the computational graph of reset during backpropagation to further improve the performance.

The hyperparameters about optimization are shown in Table B.6. We use an SGD optimizer (Rumelhart, Hinton, & Williams, 1986) with momentum 0.9 to train our models in all experiments. The weight decay is set to 1e−4 and 4e−5 for CIFAR-10/100 and ImageNet, respectively. We set the initial learning rate to 0.1 for all datasets and use a cosine learning rate decay (Loshchilov & Hutter, 2017). We adopt the mixed precision training (Micikevicius et al., 2018) in order to reduce memory consumption and speed up our training process. For the ImageNet dataset, we train our model on multi-GPU, and we use the synchronized batch normalization (SyncBN) (Zhang et al., 2018) technique.

The experiments are conducted on NVIDIA Tesla A100 GPU or NVIDIA GeForce RTX 3090 GPU.

Appendix C. Energy consumption analysis details

We conduct an energy consumption analysis to evaluate the benefit of energy efficiency of our architecture. Previous studies (Che et al., 2022; Deng et al., 2022; Lemaire et al., 2022; Li et al., 2021) have typically estimated the energy consumption of SNNs using either operational cost or memory cost. In this section, we remain consistent with these established methodologies, estimating the operational cost in Appendix C.1 and the memory cost in Appendix C.2.

Table C.7
The memory cost on CIFAR-10. "#Read" and "#Write" represent the number of read and write operations, respectively.

Architecture	#Read	#Write	Memory cost (mJ)
TANet-Tiny (ANN)	10943M	3M	109.5
TANet-Tiny (ours)	3004M	1484M	**44.9**

C.1. Operational cost

Before calculating the operational cost, we calculate the number of synaptic operations in SNNs. This calculation is performed as follows:

$$SOP^l = fr^l \times T \times FLOPs^l, \tag{C.1}$$

where l represents the layer index, SOP^l denotes the number of synaptic operations in layer l, fr^l represents the average firing rate of layer l, T denotes the number of time steps, and $FLOPs^l$ refers to the number of floating point operations of layer l.

Then, we can compute the operational cost of the convolutional and linear layers in our architecture. Due to the employment of direct encoding, the input to our architecture is non-binary and remains consistent across various time steps. Consequently, in our architecture, the first convolutional layer performs multiply-and-accumulate (MAC) operations on a single time step and duplicates its output across other time steps. In all subsequent convolutional or linear layers, the architecture transfers spikes and performs accumulate (AC) operations across all time steps. Therefore, we can quantify the operational cost of the convolutional and linear layers, denoted as E_{op}^{SNN}, as follows:

$$E_{op}^{SNN} = E_{MAC} \cdot FLOPs^1 + E_{AC} \cdot \sum_{n=2}^{N} SOP^n, \tag{C.2}$$

where N is the total number of layers, E_{MAC} and E_{AC} represent the energy cost of MAC and AC operation, respectively. $FLOPs^1$ denotes the number of floating point operations in the first layer. Refer to previous studies (Che et al., 2022; Li et al., 2021), we assume that the data for various operations are 32-bit floating-point implementation in 45 nm technology (Horowitz, 2014), with $E_{MAC} = 4.6$ pJ and $E_{AC} = 0.9$ pJ.

It is crucial to note that the BN layer is merged into the precedent convolutional layer during inference. Consequently, the BN layer incurs no operational cost during inference. Additionally, there are operations inside spiking neurons. However, it is essential to highlight that this cost is per neuron, whereas the cost for convolutional and linear layers is per synapse. Given that the number of synapses is typically thousands of times greater than that of neurons, this per-neuron cost is considered negligible. Therefore, in line with previous studies, we choose to neglect this specific cost.

C.2. Memory cost

In this section, we outline the method for estimating the memory cost of our architecture. Following the methodology in Lemaire et al. (2022), the memory cost of an SNN can be formulated as:

$$\begin{aligned} E_{mem}^{SNN} = &(RdIn^{SNN} + RdParam^{SNN} + RdPot^{SNN}) \times E_{RdRAM} \\ &+ (WrOut^{SNN} + WrPot^{SNN}) \times E_{WrRAM}, \end{aligned} \tag{C.3}$$

where $RdIn^{SNN}$, $RdParam^{SNN}$, and $RdPot^{SNN}$ represent the number of read operations for the input data, parameters, and membrane potentials, respectively. $WrOut^{SNN}$ and $WrPot^{SNN}$ denote the number of write operations to the outputs and membrane potentials, respectively. E_{RdRAM} and E_{WrRAM} represent the energy for a single read and a single write operation in RAM. In our computation, we assume $E_{RdRAM} = E_{WrRAM} = 10$ pJ.

Nevertheless, we acknowledge that relying solely on the number of operations may not provide a comprehensive evaluation of the efficiency of SNNs. A more realistic hardware evaluation is required to thoroughly evaluate the benefits of energy efficiency compared to ANNs. We recognize the importance of this and leave it to future work.

References

Albert, R., & Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, *74*(1), 47.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., & Le, Q. (2018). Understanding and simplifying one-shot architecture search. In *International conference on machine learning*.

Bu, T., Fang, W., Ding, J., Dai, P., Yu, Z., & Huang, T. (2022). Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *International conference on learning representations*.

Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2019). Once-for-all: Train one network and specialize it for efficient deployment. arXiv preprint arXiv:1908.09791.

Che, K., Leng, L., Zhang, K., Zhang, J., Meng, Q., Cheng, J., et al. (2022). Differentiable hierarchical and surrogate gradient search for spiking neural networks. In *Advances in neural information processing systems*.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Deng, S., Li, Y., Zhang, S., & Gu, S. (2022). Temporal efficient training of spiking neural network via gradient re-weighting. In *International conference on learning representations*.

DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.

Dong, X., & Yang, Y. (2019). Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Dong, X., & Yang, Y. (2020). NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International conference on learning representations*.

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., et al. (2020). SpikingJelly. https://github.com/fangwei123456/spikingjelly.

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., & Tian, Y. (2021). Deep residual learning in spiking neural networks. In *Advances in neural information processing systems*.

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., & Tian, Y. (2021). Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF international conference on computer vision*.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., et al. (2020). Single path one-shot neural architecture search with uniform sampling. In *Proceedings of the European conference on computer vision*.

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 conference on computer vision and pattern recognition*.

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*.

Horowitz, M. (2014). 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE international solid-state circuits conference digest of technical papers* (pp. 10–14). IEEE.

Kim, Y., Li, Y., Park, H., Venkatesha, Y., & Panda, P. (2022). Neural architecture search for spiking neural networks. In *Proceedings of the European conference on computer vision*.

Kim, S., Park, S., Na, B., & Yoon, S. (2020). Spiking-yolo: Spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI conference on artificial intelligence*.

Krizhevsky, A., Hinton, G., et al. (2009). *Learning multiple layers of features from tiny images*: *Tech Report*.

Kugele, A., Pfeil, T., Pfeiffer, M., & Chicca, E. (2020). Efficient processing of spatio-temporal data streams with spiking neural networks. *Frontiers in Neuroscience*, *14*, 439.

Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*, 508.

Lemaire, E., Cordone, L., Castagnetti, A., Novac, P.-E., Courtois, J., & Miramond, B. (2022). An analytical estimation of spiking neural networks energy efficiency. In *International conference on neural information processing*.

Li, Y., Guo, Y., Zhang, S., Deng, S., Hai, Y., & Gu, S. (2021). Differentiable spike: Rethinking gradient-descent for training spiking neural networks. In *Advances in neural information processing systems*.

Li, L., & Talwalkar, A. (2019). Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*.

Liu, H., Simonyan, K., & Yang, Y. (2019). DARTS: Differentiable architecture search. In *International conference on learning representations*.

Loshchilov, I., & Hutter, F. (2017). Sgdr: Stochastic gradient descent with warm restarts. In *International conference on learning representations*.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671.

Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., & Luo, Z.-Q. (2022). Training high-performance low-latency spiking neural networks by differentiation on spike representation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., et al. (2018). Mixed precision training. In *International conference on learning representations*.

Na, B., Mok, J., Park, S., Lee, D., Choe, H., & Yoon, S. (2022). AutoSNN: Towards energy-efficient spiking neural networks. In *International conference on machine learning*.

Orchard, G., Frady, E. P., Rubin, D. B. D., Sanborn, S., Shrestha, S. B., Sommer, F. T., et al. (2021). Efficient neuromorphic signal processing with loihi 2. In *2021 IEEE workshop on signal processing systems*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*.

Patel, K., Hunsberger, E., Batir, S., & Eliasmith, C. (2021). A spiking neural network for image segmentation. arXiv preprint arXiv:2106.08921.

Pham, H., Guan, M., Zoph, B., Le, Q., & Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *International conference on machine learning*.

Rathi, N., & Roy, K. (2021). DIET-SNN: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Transactions on Neural Networks and Learning Systems*.

Rathi, N., Srinivasan, G., Panda, P., & Roy, K. (2020). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *International conference on learning representations*.

Roy, K., Jaiswal, A., & Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature*, *575*(7784), 607–617.

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. *Frontiers in Neuroscience*, *11*, 682.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Sciuto, C., Yu, K., Jaggi, M., Musat, C., & Salzmann, M. (2020). Evaluating the search phase of neural architecture search. In *International conference on learning representations*.

Shrestha, S. B., & Orchard, G. (2018). Slayer: Spike layer error reassignment in time. In *Advances in neural information processing systems*.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International conference on learning representations*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, *393*(6684), 440–442.

White, C., Neiswanger, W., & Savani, Y. (2021). Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI conference on artificial intelligence*.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., et al. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Wu, Y., Deng, L., Li, G., Zhu, J., & Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, *12*, 331.

Xiao, M., Meng, Q., Zhang, Z., Wang, Y., & Lin, Z. (2021). Training feedback spiking neural networks by implicit differentiation on the equilibrium state. In *Advances in neural information processing systems*.

Xie, S., Kirillov, A., Girshick, R., & He, K. (2019). Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*.

Yan, Z., Zhou, J., & Wong, W.-F. (2021). Near lossless transfer learning for spiking neural networks. In *Proceedings of the AAAI conference on artificial intelligence*.

You, J., Leskovec, J., He, K., & Xie, S. (2020). Graph structure of neural networks. In *International conference on machine learning*.

Zenke, F., & Vogels, T. P. (2021). The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural Computation*, *33*(4), 899–925.

Zhang, H., Dana, K., Shi, J., Zhang, Z., Wang, X., Tyagi, A., et al. (2018). Context encoding for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Zheng, H., Wu, Y., Deng, L., Hu, Y., & Li, G. (2021). Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI conference on artificial intelligence*.

Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *International conference on learning representations*.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.