A Appendix

A.1 Experiments on VGG Structures

We evaluated our method on the CIFAR100 dataset using VGG architectures, besides ResNets. We treated each layer as a stage in the VGG series. During experimentation, we observed that VGG16 with three fully connected (fc) layers could not be trained effectively using the standard direct training approach (its accuracy remained limited at 1%). To tackle this issue, we merged the last three fc layers of VGG16 into one and named the resulting architecture VGG14. We set $T_{max} = 5$, $s = 3$, and computed the mean and standard deviation of three runs. We used the SGD optimizer to train the model, with a learning rate of 0.1, a weight decay of 0.0005, and a batch size of 256. The results, presented in Tab. 8, indicate the effectiveness of our approach on VGG structures.

Table 8: Accuracy of VGG on CIFAR100

Methods	Model	T=2	T=3	T=4	T=5
MTT	VGG14	73.53±0.09	74.52±0.07	75.27±0.14	75.72±0.10
InfLoR-SNN	VGG16	-	-	-	71.56±0.10

A.2 Training Details

CIFAR The CIFAR10/CIFAR100 dataset comprises 50K training images and 10K test images with a 32×32 pixel resolution. For CIFAR100, we train a ResNet-19 using the MTT pipeline for 300 epochs with a batch size of 256 and a T_{max} of 6. Following the practice in GLIF (Yao et al., 2022), the last 2 fully connected layers of ResNet-19 are replaced with a single fully connected layer. We employ the SGD optimizer with a weight decay of 0.0005 and a learning rate of 0.1 cosine decayed to 0. To make a fair comparison with the state-of-the-art (SOTA) work (Li et al., 2021b; Guo et al., 2022; Yao et al., 2022), AutoAugment (Cubuk et al., 2018) and Cutout (DeVries and Taylor, 2017) are applied to both CIFAR10 and CIFAR100 datasets. However, these augmentation techniques are only used for comparative experiments and temporal flexibility experiments, and not for other experiments.

ImageNet ImageNet (Deng et al., 2009) contains more than 1280k training images and 50k test images. We use the standard data processing flow to crop each image to a size of 224×224. We deploy the ResNet-34 structure, however, with the removal of the first max-pooling layer and changing the stride of the first basic block from 1 to 2 (Zheng et al., 2021; Yao et al., 2022). We train the model for 160 epochs with a batch size of 512 and a T_{max} of 6. We utilize the AdamW optimizer with a weight decay of 0.02 and a learning rate of 0.004 cosine decayed to 0.

DVS-Dataset CIFAR10-DVS and N-Caltech101 are neuromorphic datasets widely used in SNN experimentation. We divide the dataset into a 9:1 ratio and merge all events to form ten frames, which, similar to previous work (Li et al., 2021b; Deng et al., 2022), we resize to 48×48. For both these datasets, we adopt a random horizontal flip and rotate the frames up to 5 pixels as augmentation techniques. We employ the additional temporal inversion policy (Shen et al., 2023) uniquely for N-Caltech101. For these datasets, we use a T_{max} of 10, a batch size of 50, and train the TFSNN ResNet-18 model for 300 epochs. The optimizer we choose is SGD, with a weight decay of 0.0005, and a learning rate of 0.1, which we cosine decay to 0. While training the DVS dataset, we take only the first t frames of the ten frames where t denotes the time step of the input stage, to feed into the network.

A.3 Settings for Models on Event-driven Platform

Here, we provide details of event-driven-related experiments. We carefully learned Synsense's documentation. To obtain Spiking Neural Networks (SNNs) more suitable for deployment on asynchronous chips, we utilized a soft reset mechanism and multistep-IF neurons during training (neurons emit mem/Vth spikes per event to simulate multiple event transmissions and generations). Note that, however, the neurons realistically employed on our simulator and Speck chip are still IF neurons.

Since asynchronous event-driven chips do not support linear or convolutional layers containing biases, and networks with Batch Normalization (BN) layers inevitably introduce bias terms after absorbing BN into the convolutional layers, we adopted a more indirect method to obtain deployable networks. For 3C1FC(W16) (xCyFC(Wz) denotes VGG-like structure with x convolution layers, y fully connected layers, and maximum convolution channel z) on N-MNIST, For larger and deeper networks like 4C2FC(W32) for DVS-Gesture and VGGSNN for CIFAR10-DVS, we first train with networks that include BN layers to establish a baseline model, then absorb BN into the convolutional layers, remove the bias, and further fine-tune to achieve a network without bias terms.

For NMNIST, our final layer is an IF voting layer; for DVS-Gesture and CIFAR10-DVS, we find it crucial to replace the final layer with a membrane potential voting layer.

A.4 Design TTM Grouping Policy

We previously stated our policy's objective is to partition l frames into k groups ($l \geq k$) as evenly as possible. In this section, we mathematically interpret the design. We will start by describing the grouping process in a different manner. The l frames are viewed as l adjacent intervals of length 1 over the rational number domain with the i-th frame starting at $i-1$ and ending at i. We define c_i as the boundary between group i and group $i-1$. Here, i ranges from 1 to k, and c_1 is 0. Ideally, $c_i = (i-1) \cdot l/k$ is set to group frames most evenly. Nevertheless, this strategy produces non-integer c_i, which results in atomic frames' division when l is not a multiple of k. To solve this problem, we retreat and set c_i to the nearest integer and get

$$c_i = \lfloor \frac{(i-1) \cdot l}{k} - \varepsilon \rceil, \tag{12}$$

where ε is a small constant used to determine c_i when the distances to the closest two integers are equal. As b_i must be the frame directly following the boundary c_i ($b_i = c_i + 1$), we obtain Eq. 8.

A.5 Derivation of the Backpropagation Formula for LIF

In this section, we derive the backpropagation formula for LIF from the forwarding formula. We derive $\partial u(t)/\partial v(t)$ from Eq. 4 first:

$$\frac{\partial u(t)}{\partial v(t)} = 1 - s(t) - v(t) \cdot \frac{\partial s(t)}{\partial v(t)}. \tag{13}$$

Then, we consider the derivation of $\partial u(t)/\partial v(t-1)$. According to Fig. **??**, $\partial u(t)/\partial v(t-1)$ can be calculated as follows

$$\frac{\partial u(t)}{\partial v(t-1)} = \frac{\partial u(t)}{\partial v(t)} \frac{\partial v(t)}{\partial u(t-1)} \frac{\partial u(t-1)}{\partial v(t-1)} = \tau \frac{\partial u(t)}{\partial v(t)} \frac{\partial u(t-1)}{\partial v(t-1)}. \tag{14}$$

By combining multiple Eq. 14, we get

$$\frac{\partial u(t)}{\partial v(t-n)} = \tau^n \prod_{i=t-n}^{t} \frac{\partial u(i)}{\partial v(i)}. \tag{15}$$

Finally, we get the complete expression for $\partial s(t)/\partial I(t-n)$ as follows

$$\begin{aligned} \frac{\partial s(t)}{\partial I(t-n)} &= \frac{\partial s(t)}{\partial v(t)} \frac{\partial v(t)}{\partial u(t-1)} \frac{\partial u(t-1)}{\partial v(t-n)} \frac{\partial v(t-n)}{\partial I(t-n)} \\ &= \frac{\partial s(t)}{\partial v(t)} \cdot \tau^n \prod_{i=t-n}^{t-1} [(1 - s(i)) - v(i) \cdot \frac{\partial s(i)}{\partial v(i)}] \end{aligned} \tag{16}$$

A.6 T = T$_{\text{MAX}}$ BN Statistics vs. Recalculated BN Statistics

As previously mentioned, we discovered that the BN statistics of the T=T$_{\text{max}}$ network can be applied to other TFSNN with uniform T across blocks. In this section, we provide experimental verification for this observation. Our experiment involves testing the accuracy of one of our trained ResNet-19 models with two distinct BN layer information approaches. The first approach utilizes the statistics of the T=T$_{\text{max}}$ network, while the second approach recalculates the BN statistics individually for each time step T. For the latter approach, we calibrate the BN layer three times and report the average accuracy. Our experimental results demonstrated in Table 10 show that the mean and variance calculated at T=T$_{\text{max}}$ is applicable directly to other T values. Therefore, we can utilize the BN statistics of T=T$_{\text{max}}$ for other T directly, which saves the time required to calibrate the BN layers for other T values.

Table 9: Accuracy given sampling number s and training epochs e.

Sampling Num	$e \times s$			
	300	600	900	1200
$s = 1$	**75.61**	76.13	76.23	75.78
$s = 2$	75.53	76.37	76.31	76.36
$s = 3$	75.25	**76.45**	**76.47**	**76.44**
$s = 4$	74.81	75.74	76.41	76.42

Table 10: Test accuracy of a single model with two kinds of BN statistics.

Method	TimeStep			
	2	3	4	5
T=$T_{\max}$ stat	80.21	81.06	**81.51**	81.82
Recalculated stat	80.21	**81.23**	81.44	**81.85**

A.7 Impact of Different Sampling Number and Training Epochs

In most previous experiments, we employed sampling number $s = 3$. In this section, we experiment with varying values of s, assess their effects at different epochs, and explain why we chose $s = 3$. We train ResNet-18s with $T_{max} = 6$ on CIFAR100 with varied s and list their test accuracy at T=6. The results are displayed in Table 9. When $e \times s$ is constrained, the model requires more epochs to converge, necessitating a lower s. However, if trained across a sufficient number of epochs, sampling of s structurally diverse networks can smooth the optimization of network parameters and improve performance. Specifically, we find $s = 3$ performs well and adopt $s = 3$ for most of the experiments.

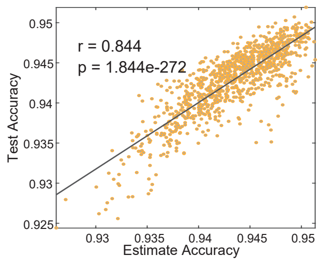

Figure 7: Correlation curve for estimate accuracy and test accuracy on CIFAR10

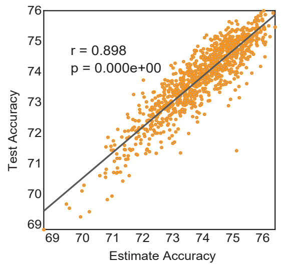

Figure 8: Correlation curve for estimate accuracy and test accuracy on CIFAR100

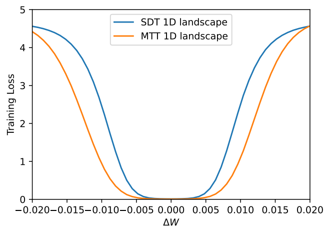

Figure 9: The 1D landscapes of ResNet18 trained by SDT and MTT on CIFAR100.

A.8 Partitioned SNN Accuracy Estimation for Different Temporal Configurations on CIFAR10/100

As MTT divides SNN into stages of different T when training, we are interested in the performance of each possible temporal configuration $\boldsymbol{t}$. Although the stage-wise different time steps are only applied to the MTT training process to formulate a better TFSNN and are not necessary for inference, the relationship between $\boldsymbol{t}$ and inference performance can inspire SNN structure designs. However, it is impossible to test all possible situations directly (e.g., there are 6^8 cases for ResNet-18). Therefore, we propose the following hypotheses to estimate the accuracy of each $\boldsymbol{t}$: 1) The expressive power of each block in TFSNN contributes differently and positively to the final network accuracy. 2) The expressive power of each block is related to the information content of its selected time T, such as $K\sqrt{\log_2 T}$, where the square root is due to the information content of a spike sequence with time step T cannot exceed $\log_2 T$ because the arrangement of spikes is regular, e.g., tending to be uniformly distributed. Then we assume that the accuracy equation of different temporal configurations is $\sum_{i=1}^{I} K_i\sqrt{\log_2 T_i} + c$, where the K_i is the contribution of each block, and c is a constant. Our experiment demonstrates that we only need to sample a very small number of $\boldsymbol{t}$s to infer the parameters of the equation, and then able to estimate all TFSNN accuracy with this equation.

Here, we randomly sample 18 temporal configuration vectors (different time step combinations) and obtain their test accuracy for solving the hypothesis equation in Sec. A.8. The weight parameters we obtained are $\{0.93, 0.53, 0.59, 0.67, 1.22, 0.48, 1.36, 0.18\}$, and the constant value c is 67.11. This result supports our hypothesis 1), which suggests that all blocks' time step increment positively contributes to the accuracy of the final network. Some blocks, such as blocks 1, 5, and 7, have a greater contribution. Then, we resample 1000 configurations and validate their estimated accuracy and their test accuracy. The result (Fig. 6 (A)) shows that our method can effectively predict the actual testing accuracy of different temporal configurations. Finally, we use the spike frequency and accuracy estimation to build a combinatorial optimization equation for searching the optimal configuration (see Sec A.9 for detail). For example, by setting the energy cost that is lower than default (the time step of all blocks is 3), we discover the optimal combination of block time steps is $\{3, 2, 2, 3, 5, 3, 6, 2\}$. The selected configuration acquires an accuracy of 75.38%, which is 0.62% higher than the default.

In addition to CIFAR100, we also conducted experiments with ResNet-18 on CIFAR10/100. We randomly sample 18 temporal configurations as usual and solve the equation in Sec. A.9. The weight parameters are $\{0.0049, 0.0028, 0.0017, 0.0037, 0.0037, 0.0004, 0.0005, 0.0017\}$, and the constant value c is 92.13. Notes that block 1,4,5 have a higher contribution, and the 1,5 blocks are also highly weighted on CIFAR100, which may imply that the weights are partly related to the network structure. We then resample 1000 temporal configurations and plot their estimate accuracy and test accuracy on CIFAR10 in Fig. 7 and CIFAR100 in Fig.8, respectively. We also use the aforementioned combinatorial optimization strategy to search the optimal temporal configuration under the energy consumption of T=3 and find $\{4, 2, 2, 3, 5, 2, 2, 4\}$, which achieves an accuracy of 94.70%, 0.21% higher than its T=3 counterpart.

A.9 Details of Combinatorial Optimization

As previously mentioned, the accuracy formula of different temporal structures is given by the expression $\sum_{i=1}^{I} K_i \sqrt{\log_2 t_i} + c$, where K_i represents the contribution of each block, I is the number of blocks, and c is a bias. We randomly select 18 distinct temporal configurations ($\boldsymbol{t}$) and evaluate their accuracies on the test set, resulting in 18 pairs of temporal configurations and their corresponding accuracies. Using the least squares method, we compute the values of K_i and c from the collected data. Next, we estimate the average firing rate (R_i) of each block in a unified SNN of T=6. Then, the energy consumption of a specified temporal configuration t can be approximated as $\sum_{i=1}^{I} t_i \cdot R_i$. For example, the estimated energy consumption of a unified SNN with T=4 is calculated as $EC_4 = \sum_{i=1}^{I} 4R_i$. Based on this, we can obtain a group of TFSNNs with lower energy consumption (EC) for a given T=T$_g$, and we aim to identify the TFSNN with the maximum estimated accuracy from this set. This is formulated as the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \text{ACC}_{\text{estimated}} = \sum_{i=1}^{I} K_i \sqrt{\log_2 t_i} + c \\ \text{s.t.} \quad & \sum_{i=1}^{I} t_i \cdot R_i \leq EC_{T_g} \\ & T_{min} \leq t_1, t_2, \ldots, t_l \leq T_{max}, \end{aligned}$$

where $t_i$is the i-th component of temporal configurations $\boldsymbol{t}$ and EC_{T_g} is the given uper bound of the TFSNN energy.

In order to solve the above problem, we adopt the depth-first search (DFS) algorithm to search in the solution space. To obtain a more accurate accuracy for each temporal configuration $\boldsymbol{t}$, we perform three times of BN calibrations and take the average of the accuracies.

A.10 Loss Landscapes of MTT and SDT

To visually confirm the flatter minimum achieved by the model trained with MTT, we trained ResNet18 using SDT and MTT on CIFAR100 and plotted their loss landscapes in Fig. 9. We observed that MTT led the model to a flatter minimum which indicates improved generalizability.

A.11 Verifying Generalizability Through Gradient Metrics

Apart from noise injection, another famous metric that indicates the generalizability is the length of the gradient on weights $||\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}||$ and the inputs $||\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_i}||$. For $||\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}||$, we evaluate the length of the gradient of loss over the entire training set for the convolution layers. For $||\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_i}||$, we calculate the mean value of the length of each input gradient. The model trained by MTT exhibits a shorter gradient of both weights and inputs (see Tab. 11), which implies the model's strong robustness and generalizability.

Table 11: The gradient statistics of the model trained by SDT and MTT.

Methods	$\|\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}\|\|$	$\|\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_i}\|\|$
MTT	11.59	1.81
SDT	38.08	7.78

Table 12: Results on seqMNIST.

Methods	**Acc**
Our RNN	56.22
SNN SDT	55.75
SNN MTT	**64.56**

Table 13: Results on Spiking Heidelberg Digits

Methods	**Acc**
l=3 Repro by code of Hammouamri et al. (2023)	75.26
l=3 our SDT	74.43
l=3 our MTT	**79.68**

Table 14: Results on Spiking Speech Commands

Methods	**Acc**
Our SDT l=3	57.75
Our MTT l=3	**60.15**

A.12 Experiments on Audio and Sequential Datasets

Our research reveals that models trained by MTT can function effectively as a time encoder when the temporal configuration vectors used for training are monotonically non-increasing.

To illustrate this adaptability, we present the performance of TFSNN on three distinct temporal tasks: seqMNIST, Spiking Heidelberg Digits and Spiking Speech Commands.

For the sequential task seqMNIST, we utilized a simple fc LIF SNN with 2 hidden layers of width 64 and set the time constant τ to 0.99. We also trained an RNN with 2 hidden layers of width 64 for comparison. The results are as shown in Tab. 12.

For the Spiking Heidelberg Digits, we adopt the plain 3-layer feed-forward SNN architecture proposed by Cramer et al. (2020), a fully connected SNN with an input width of 70 and 128 LIF neurons in each of the 3 hidden layers. The timestep of the first layer is fixed to the input timestep, while the timesteps of subsequent layers are restricted to monotonically non-increasing. We set $\tau = 0.9753$, which is equivalent to the parameter λ in the work of Cramer et al. (2020), namely $1 - 1/\tau$ in most other articles, and train the model for 150 epochs. To ensure the validity of our results, we also reproduce the result using the code provided by Hammouamri et al. (2023). The results are as shown in Tab. 13.

Spiking Speech Commands (SSC) (Cramer et al., 2020) is a spiking dataset converted from Google Speech Commands v0.2 and is tailored for SNN. For SSC, we continue using the same architecture and the same parameters as we used in SHD, except that here we only train the model for 60 epochs. The results are shown in Tab. 14.