

TT-SNN: Tensor Train Decomposition for Efficient Spiking Neural Network Training

Donghyun Lee, Ruokai Yin, Youngeun Kim, Abhishek Moitra, Yuhang Li, Priyadarshini Panda
Department of Electrical Engineering, Yale University, USA
{donghyun.lee, ruokai.yin, youngeun.kim, abhishek.moitra, yuhang.li, priya.panda}@yale.edu

Abstract—Spiking Neural Networks (SNNs) have gained significant attention as a potentially energy-efficient alternative for standard neural networks with their sparse binary activation. However, SNNs suffer from memory and computation overhead due to spatio-temporal dynamics and multiple backpropagation computations across timesteps during training. To address this issue, we introduce Tensor Train Decomposition for Spiking Neural Networks (TT-SNN), a method that reduces model size through trainable weight decomposition, resulting in reduced storage, FLOPs, and latency. In addition, we propose a parallel computation pipeline as an alternative to the typical sequential tensor computation, which can be flexibly integrated into various existing SNN architectures. To the best of our knowledge, this is the first of its kind application of tensor decomposition in SNNs. We validate our method using both static and dynamic datasets, CIFAR10/100 and N-Caltech101, respectively. We also propose a TT-SNN-tailored training accelerator to fully harness the parallelism in TT-SNN. Our results demonstrate substantial reductions in parameter size (7.98 \times), FLOPs (9.25 \times), training time (17.7%), and training energy (28.3%) during training for the N-Caltech101 dataset, with negligible accuracy degradation.

Index Terms—Neuromorphic computing, Spiking neural network, Tensor train decomposition

I. INTRODUCTION

Spiking Neural Networks (SNNs) have gained significant interest as a low-power substitute to Artificial Neural Networks (ANNs) in the past decade [1]. Unlike ANNs, SNNs process visual data in an event-driven manner, employing sparse binary spikes across multiple timesteps. This unique spike-driven processing mechanism brings high energy efficiency on various computing platforms [2], [3]. To leverage the energy-efficiency advantages of SNNs, many SNN training algorithms have been proposed, which can be categorized into two approaches: ANN-to-SNN conversion [4], [5] and backpropagation (BP) with surrogate gradient [6], [7]. Among them, BP-based training stands out as a mainstream training method as it not only achieves state-of-the-art performance but also requires a small number of timesteps (≤ 5). However, as BP-based training computes backward gradients across multiple timesteps and layers, SNNs require substantial training memory to store the intermediate activations [8].

To address the challenge, various efficient techniques have been applied to SNNs, including quantization [9], [10], Knowledge Distillation (KD) [11], [12], and pruning [8]. In [9], the authors focus on quantizing trainable weights to enable efficient and faster inference on SNN architecture. Additionally, prior

research [10] has explored the weight and membrane potential quantization to 2-bit for efficient hardware implementation. In terms of KD, in [11], the ANN-based teacher model transfers the knowledge to the student model which is an SNN architecture for faster convergence. In contrast, in [12], the teacher model is SNN architecture, whose spike distribution is transferred to stabilize the training process of the student model. These efforts have proven successful in reducing memory costs and the total number of timesteps required, all while achieving specific accuracy targets. Nonetheless, the previous techniques mostly aim at fast and light inference, rather than focusing on training efficiency.

In this work, we introduce the TT-SNN module for accelerating training in SNNs by applying Tensor Train (TT) decomposition [13]. Our approach involves decomposing the weights of convolutional layers into smaller tensors, resulting in a lighter and faster training process. Inspired by [14], we modify the computation pipeline with parallel asymmetric-sized kernels, which we term as Parallel TT (PTT), in contrast to the conventional Sequential TT (STT) operations. Additionally, we introduce the Half TT (HTT) module, which employs partial parts of PTT computations to boost training efficiency. Following the training process, we reconstruct the decomposed convolutional weights to maintain spike-inspired computation in the inference pipeline.

From the hardware perspective, the recent advancements in SNN-tailored training accelerators, such as [3] and [15], have paved the way for efficient SNN training. Given that the STT method operates on a single sub-convolutional layer at a time, it aligns well with the design of existing SNN training accelerators [3], [15], where each sub-convolutional layer is mapped sequentially onto the computation engines of the accelerator. This ensures a direct and efficient mapping without the need for significant architectural modifications. However, the PTT and HTT methods introduce a level of parallelism that is not optimally handled by the existing SNN training accelerators. The reason is that prior accelerators are primarily designed for single-layer workloads. The parallelism inherent in PTT and HTT, where two sub-convolutional layers operate concurrently, demands a more intricate hardware design. In this work, we propose a TT-SNN-tailored training accelerator design to fully harness the parallelism from the proposed PTT and HTT methods.

The main contributions of our work are as follows: (1) We propose the TT-SNN module, which leverages TT decomposi-

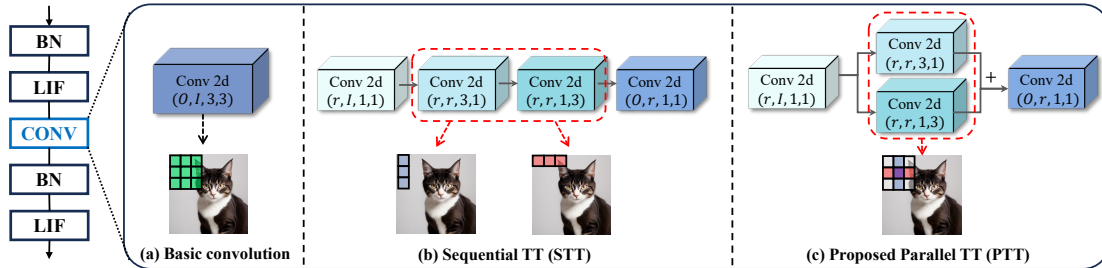


Fig. 1: Illustration of TT-SNN modules. The order of weight information is followed according to the pytorch framework, i.e., (output channel, input channel, kernel size, kernel size) (a) Basic convolution weights with 3×3 kernel. (b) Sequential computation of TT-cores is considered a traditional method with asymmetric kernels. (c) Proposed Parallel TT-module (PTT). Two asymmetric kernels are computed in parallel with the output of the first sub-convolution. The parallel computation of PTT can be seen as 3×3 without the four corner values.

tion to enhance the efficiency of SNN architecture by enabling faster computation and reducing memory costs during training. Departing from the typical sequential computations, we introduce parallel processing into the training pipeline. (2) The TT-SNN module can be easily and flexibly integrated into SNN convolutional computations. (3) We propose a multi-cluster systolic-array-based SNN training accelerator to efficiently implement and evaluate the TT-SNN-based training. Compared to the existing SNN training accelerator, our design further reduces 28.3%(43.5%) energy cost for PTT(HTT) training. (4) Our experiments demonstrate that TT-SNN reduces the number of trainable parameters, floating-point operations (FLOPs), and training time on datasets such as CIFAR10/100 and N-Caltech101 [16] without significant accuracy loss. For example, on the N-Caltech101 dataset, TT-SNN achieves compression ratios of $7.98 \times$ in parameters, $9.25 \times$ in FLOPs, and 17.66% training time reduction, which highlights its compatibility with event datasets.

II. PRELIMINARY

Spiking Neural Network: SNNs are brain-inspired architectures designed for efficient computation on neuromorphic devices. It relies on the Leaky-Integrate-and-Fire (LIF) neuron [17], a non-linear function that closely mimics biological neurons in humans, making it an ideal choice for SNN design due to its efficient computation. We use the iterative LIF neuron model in [6] for designing SNN architecture as follows:

$$u_i^{l,t} = \tau_m u_i^{l,t-1} + \sum_{j=1}^n w_{ij} H(u_i^{l,t} - V_{th}), \quad (1)$$

where, $u_i^{l,t}$ is the membrane potential of i -th neuron in l -th layer at timestep t , $\tau_m \in (0, 1]$ is the leaky factor for membrane potential leakage, $H(\cdot)$ is the Heaviside step function with firing threshold V_{th} . When a spike fires, $u_i^{l,t} \geq V_{th}$, the membrane potential $u_i^{l,t}$ is reset to 0.

Tensor Train Decomposition: Tensor decomposition has emerged as a promising compression technique for mitigating the high redundancy inherent in Deep Neural Networks (DNNs) by reducing the number of trainable parameters. There are primarily three distinct types of tensor decomposition techniques applied to DNNs [18], i.e., CANDECOMP/PARAFAC (CP),

Tucker, and TT decomposition. While CP and Tucker decompositions are compact and efficient, they may not be suitable for large-scale models due to the curse of dimensionality [19]. In contrast, TT decomposition is less sensitive to the curse of dimensionality and remains a powerful tool for reducing the number of trainable parameters, that can be expressed as

$$\mathcal{A} = \mathcal{G}_1 \times^1 \mathcal{G}_2 \times^1 \cdots \times^1 \mathcal{G}_d, \quad (2)$$

where $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ is a d -dimensional target tensor, $\mathcal{G}_k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ ($k \in \{1, 2, \dots, d\}$) is the k -th decomposed tensor, and \times^1 denotes the contraction. \mathcal{G}_k is called as TT-core and r_k ($r_0 = r_d = 1$) is TT-rank which controls the complexity of decomposition. While TT decomposition has been applied to several DNN structures [20], [21], most of these applications mainly concentrate on reducing theoretical computation complexity rather than addressing actual hardware latency. In contrast, Gabor et al. [22] successfully reduce FLOPs in each CNN layer by incorporating circular permute into the TT decomposition.

$$\mathcal{W} = \text{circular_permute}(W, -1) \in \mathbb{R}^{I \times K \times K \times O} \quad (3)$$

Here, W refers to CNN weights, I is the number of input channels, O is the number of output channels, and K is kernel size. Then, \mathcal{W} can be decomposed by TT-format according to Eq.(2), which can be represented as

$$W_{I,K_1,K_2,O} = \sum_{r_1}^{R_1} \sum_{r_2}^{R_2} \sum_{r_3}^{R_3} w_{I,r_1}^{(1)} w_{r_1,K_1,r_2}^{(2)} w_{r_2,K_2,r_3}^{(3)} w_{r_3,O}^{(4)}. \quad (4)$$

Based on [22], one convolution computation can be separated into four sub-convolutions with smaller weights like Fig. 1(b), which results in a faster and lighter training process. This approach is different from the TT decomposition used in previous works [20], [21], where an extra reconstruction process of TT-cores is applied in every convolutional operation during both training and inference.

III. PROPOSED METHOD

In this section, we begin by introducing the TT-SNN framework, which incorporates TT decomposition into SNNs. The first TT module is the PTT, designed to parallelize convolutional computations. Additionally, we present another TT

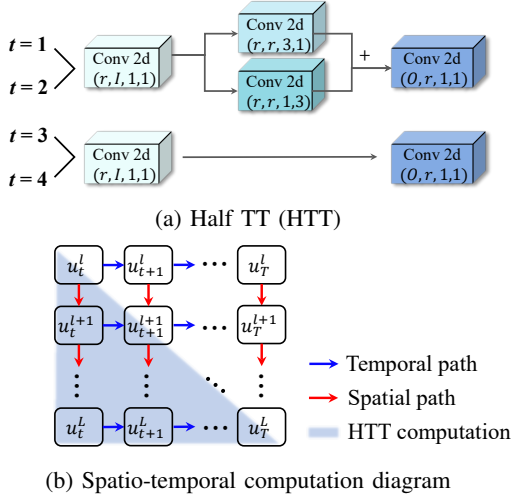


Fig. 2: Illustration of Half TT (HTT) format for further compression. (a) Instead of sharing all weights through timesteps, HTT uses partial parts of sub-convolutions. (b) In the spatio-temporal computation dimension of SNN, the HTT module takes up a half-diagonal area due to its partial usage of weights through timestep.

module, HTT, specifically engineered to enable half-diagonal computation within the spatio-temporal computation dimension. Finally, we demonstrate the overall training process of TT-SNN architecture.

Parallel TT Module: To address the challenges posed by the high training complexity of SNNs, as discussed in Section I, we introduce a novel and straightforward training pipeline for SNNs called TT-SNN, illustrated in Fig. 1. TT decomposition allows us to break down one convolution into four sub-convolutions. In [22], they sequentially compute the sub-convolutions, like Fig. 1(b), resulting in considerable performance with a reduction in the number of parameters and FLOPs. However, the Sequential TT (STT) loses the input information due to its asymmetric kernel size. The second and third sub-convolutions in STT compute with $(3, 1)$ and $(1, 3)$ kernels, leading to either vertical or horizontal information extraction. Consequently, the perpendicular direction information of each kernel from the previous layer is overlooked.

In order to address this asymmetry in STT, we propose a simple modification, called Parallel TT (PTT) illustrated in Fig. 1(c). In the PTT pipeline, the second and third sub-convolution layers are computed in parallel, both utilizing the output from the first sub-convolution layer. This parallel computation in PTT resembles a 3×3 kernel without the four corner values. Hence, this cross-sectional kernel can perceive the vertical and horizontal feature information simultaneously, resulting in improved performance. Motivated by [14], we restructure the computations of sub-convolutions using Eq. (4) as follows:

$$y_t = [(x_t * w_{I,r}^{(1)} * w_{r,K_1,r}^{(2)}) + (x_t * w_{I,r}^{(1)} * w_{r,K_2,r}^{(3)})] * w_{r,O}^{(4)}, \quad (5)$$

where x_t, y_t are input and output in timestep t respectively, and $*$ denotes convolution computation.

Half TT Module: In traditional SNN architectures, weight sharing across timesteps is a common approach to keep weight

storage consistent, even as the number of timesteps increases. [23] finds that SNNs tend to capture more information during the early timesteps compared to the later ones, implying the existence of redundancy in timesteps. To explore this argument, we introduce a novel strategy called the Half TT (HTT) module, which operates by using only half of the sub-convolutions in select timesteps, as opposed to employing all sub-convolutions throughout all timesteps as depicted in Fig. 2(a). The main difference between PTT and HTT is that PTT employs all sub-convolutions throughout the entire timestep, while HTT uses half of the sub-convolutions in specific timesteps. We place full sub-convolutions in the early timesteps and reserve half sub-convolutions for the later timesteps. In this approach, we can weaken the timestep redundancy and achieve faster and more resource-efficient computation. The HTT can be seen as a half-diagonal computation in a spatio-temporal computation graph diagram on SNN shown in Fig. 2(b).

Training Pipeline: By aggregating the proposed TT modules, we represent an efficient training pipeline, as outlined in Algorithm 1. To begin, we initialize the base SNN model to gain optimized TT-ranks using Variational Bayesian Matrix Factorization (VBMF) [24]. VBMF is a practical tool for estimating near-optimal ranks with automatic posterior approximation. Subsequently, the TT-SNN model is initialized with decomposed weights and the acquired TT-ranks (lines 1-5). Note that the first CNN layer and the last classifier are not decomposed layers as customization of these layers results in a significant drop in accuracy. After training (lines 6-18), the entire weights of sub-convolutions are merged into a single original weight to enable spike-based computations throughout the model (lines 19-21). The Eq. (6) shows the reconstruction process from Eq. (5). We simplify the weight terms in Eq. (5) from $\{w_{I,r_1}^{(1)}, w_{r_1,K_1,r_2}^{(2)}, w_{r_2,K_2,r_3}^{(3)}, w_{r_3,O}^{(4)}\}$ to $\{w^{(1)}, w^{(2)}, w^{(3)}, w^{(4)}\}$.

$$\begin{aligned} y_t &= [(x_t * w^{(1)} * w^{(2)}) + (x_t * w^{(1)} * w^{(3)})] * w^{(4)} \\ &= [x_t * (w^{(1)} \times^1 w^{(2)}) + x_t * (w^{(1)} \times^1 w^{(3)})] * w^{(4)} \\ &= x_t * (w^{(1)} \times^1 w^{(2)} \times^1 w^{(4)} + w^{(1)} \times^1 w^{(3)} \times^1 w^{(4)}) \\ &= x_t * \widetilde{W}. \end{aligned} \quad (6)$$

In summary, our proposed approach offers substantial advantages during training, and pre-trained TT modules can be converted into the base model architecture without incurring any significant losses.

IV. SNN TRAINING ACCELERATOR DESIGN FOR TT-SNN

To fully harness the parallelization from the PTT and HTT methods, we propose a multi-cluster systolic-array-based SNN training accelerator design as shown in Fig. 3. In our design, we have 4 computation clusters for mapping the workload of each sub-convolutional layer. The cluster ① computes the first sub-convolutional layer. Since the input is in the form of spikes, we simplified the arithmetic units inside the PEs of ①. As shown in Fig. 3, cluster ② and cluster ③ run in parallel (shown in the red rectangle). The outputs from ① will first be written into an output buffer and then consumed by the clusters ② and ③ and

Algorithm 1 Training process of TT-SNN

l, t , FC, LIF, and BN represent the l -th layer and the t -th timestep, the fully-connected layer, the LIF neuron model in Eq. (1), and the batch normalization respectively

```

1:  $[\mathcal{W}_1, \mathcal{W}_2, \dots, \mathcal{W}_L] = \text{Initialize}(\text{base model})$ 
2:  $[r_2, r_3, \dots, r_{L-1}] = \text{VBMF}([\mathcal{W}_2, \mathcal{W}_3, \dots, \mathcal{W}_{L-1}])$ 
3: for  $l \leftarrow 2$  to  $L-1$  do
4:    $[w_l^{(1)}, w_l^{(2)}, w_l^{(3)}, w_l^{(4)}] = \text{Initialize}(\mathcal{W}_l, r_l)$ 
5: end for
6: for epochs do
7:   for  $t \leftarrow 1$  to  $T$  do
8:      $y_{t,1} = x_{t,1} * \mathcal{W}_1$ 
9:     for  $l \leftarrow 2$  to  $L-1$  do
10:       $x_{t,l} = \text{BN}(\text{LIF}(y_{t,l-1}))$ 
11:       $o_{t,l} = x_{t,l} * w_l^{(1)}$ 
12:       $y_{t,l} = [(o_{t,l} * w_l^{(2)}) + (o_{t,l} * w_l^{(3)})] * w_l^{(4)}$ 
13:    end for
14:     $y_{t,L} = \text{FC}(\text{LIF}(y_{t,L-1}))$ 
15:  end for
16:   $L = \text{Cross-Entropy}(\sum_{t=1}^T y_{t,L}, \text{label})$ 
17:  Calculate  $\frac{\partial L}{\partial w_l^{(1)}}, \frac{\partial L}{\partial w_l^{(2)}}, \frac{\partial L}{\partial w_l^{(3)}}, \frac{\partial L}{\partial w_l^{(4)}}$ 
18:  Update  $w_l^{(1)}, w_l^{(2)}, w_l^{(3)}, w_l^{(4)}$ 
19: end for
20: for  $l \leftarrow 2$  to  $L-1$  do
21:    $\tilde{\mathcal{W}}_l = (w_l^{(1)} \times^1 w_l^{(2)} \times^1 w_l^{(4)}) + (w_l^{(1)} \times^1 w_l^{(3)} \times^1 w_l^{(4)})$ 
22: end for

```

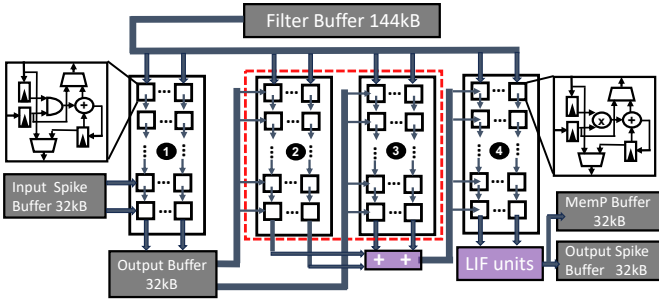


Fig. 3: Illustration of the design of our training accelerator for efficiently mapping the PTT-SNN and HTT-SNN. MemP denotes the membrane potential.

③. The generated results are then merged in the adder array units and sent to the cluster ④ for the computation of the last sub-convolutional layer. To support the non-spike inputs to those layers, we equip the PEs with multipliers in those three clusters ② - ④. The outputs from the last cluster will be sent to the LIF array units to be converted back to forms of spikes. Finally, both the spikes and the membrane potentials from the LIF units will be written back to the corresponding global buffers. We run the whole design in a highly pipelined fashion. While writing the inputs and weights to ①, the weights are also being filled to the cluster ② and ③ to hide the SRAM read latency. The outputs from the global output buffer after the cluster ① are instantly consumed by the parallel clusters ②, ③. The results from the adder arrays are also instantly consumed by the cluster ④.

We adopt three levels of the memory hierarchy: 1) an off-chip DRAM, 2) SRAM-based global buffers, and 3) small register-file-based scratch pads. The output-stationary dataflow is adopted in cluster ① and ④ and the weight-stationary dataflow is adopted in cluster ② and ③ for matching the latency between clusters. We will finish processing all timesteps

for each layer and then move to the next [25]. We follow the dataflow in [3] and [15] to accelerate the BPTT-based backpropagation. The detailed design configurations of our design can be found in Table. I.

TABLE I: Hardware Implementation Parameters.

| Technology | 28nm CMOS |
|--------------------------|-----------|
| # of Cluster | 4 |
| # of PE / Cluster | 32 |
| Scratch Pad Size / PE | 32 bytes |
| Total Global Buffer Size | 272 KB |
| Accumulator Precision | 16-bits |
| Multiplier Precision | 8-bits |

V. EXPERIMENTS

A. Implementation Details

Software: We evaluate our work on CIFAR10/100 with ResNet18 and N-Caltech101 with ResNet34 architecture, respectively. We have adopted MS-ResNet [30] as our baseline SNN architecture. We use direct coding to convert a float pixel value into binary spikes [31]. During the training process, we use SGD optimizer with momentum 0.9 and weight decay $1e-4$. We adopt cosine annealing scheduler for learning rate decay with initial learning rate 0.1. We set the number of epochs as 100 for all datasets. The batch size for CIFAR10, CIFAR100, and N-Caltech101 is set to 100, 100, and 50 respectively. In terms of spiking mechanism, we set τ_m and V_{th} to 0.25, 0.5 respectively in Eq. (1). The TT-ranks attained by VBMF for ResNet18 is $\{24, 27, 25, 29, 37, 45, 43, 41, 65, 74, 70, 63, 104, 153, 186, 145\}$, and for ResNet34 is $\{24, 23, 22, 17, 16, 12, 22, 31, 25, 25, 24, 21, 20, 19, 48, 79, 64, 69, 63, 69, 60, 65, 63, 63, 62, 58, 121, 170, 173, 147, 161, 108\}$. The training timestep is 4 for CIFAR10 and 6 for N-Caltech101. When we use the HTT module for training CIFAR10/100 and N-Caltech101, half sub-convolutions are applied in timestep $t = 3, 4$ and $t = 5, 6$ respectively. All experiments are conducted by RTX 3090ti GPUs.

Hardware: We synthesize our accelerator in Sec. IV using Synopsys Design Compiler at 400MHz using 28nm CMOS technology. We use CACTI to simulate on-chip SRAM and off-chip DRAM to obtain memory statistics. The energy results are generated from SATASim, a cycle-accurate SNN training energy simulator [3]. The training energy includes the computation and the data movement cost for both the forward and the backward propagation of one image across all timesteps.

B. Experimental Results

We have summarized the main results for the CIFAR10/100, N-Caltech101 datasets in Table II, which includes accuracy, training time, the number of trainable parameters, FLOPs. Note that the training time denotes the time taken for forward and backward passes on a single batch of inputs of a given dataset. **CIFAR10/100:** The results for CIFAR10/100 exhibit very similar trends across all evaluation metrics. As anticipated, among the TT modules, PTT achieves the highest accuracy but also an insignificant accuracy drop compared to the baseline. In terms of training time, PTT significantly reduces the training time, approximately 17% faster than the baseline, and outperforms

TABLE II: Results on CIFAR10/100 with ResNet18 and N-Caltech101 with ResNet34 architecture. All metrics are computed during the training process. Here, training time represents the time taken for forward and backward passes on a single batch of inputs, M and G denote millions and gigabytes respectively.

| Dataset | Method | Accuracy (%) | Training time (s) | # of parameters (M) | FLOPs (G) |
|-------------------------|----------|--------------|-------------------|---------------------|----------------|
| CIFAR10 (t = 4) | baseline | 93.41 | 0.214 | 11.20 | 2.221 |
| | STT | 90.91 | 0.190 (11.21 % ↓) | 1.83 (6.13 ×) | 0.372 (5.97 ×) |
| | PTT | 91.65 | 0.176 (17.76 % ↓) | 1.83 (6.13 ×) | 0.372 (5.97 ×) |
| | HTT | 91.19 | 0.166 (22.43 % ↓) | 1.83 (6.13 ×) | 0.282 (7.88 ×) |
| CIFAR100 (t = 4) | baseline | 72.49 | 0.214 | 11.21 | 2.222 |
| | STT | 68.49 | 0.190 (11.21 % ↓) | 1.69 (6.62 ×) | 0.373 (5.96 ×) |
| | PTT | 70.44 | 0.176 (17.76 % ↓) | 1.69 (6.62 ×) | 0.373 (5.96 ×) |
| | HTT | 70.22 | 0.167 (21.96 % ↓) | 1.69 (6.62 ×) | 0.282 (7.87 ×) |
| N-Caltech101 (t = 6) | baseline | 77.13 | 0.657 | 21.31 | 15.65 |
| | STT | 76.48 | 0.572 (12.94 % ↓) | 2.67 (7.98 ×) | 1.69 (9.25 ×) |
| | PTT | 77.24 | 0.541 (17.66 % ↓) | 2.67 (7.98 ×) | 1.69 (9.25 ×) |
| | HTT | 75.38 | 0.530 (19.33 % ↓) | 2.67 (7.98 ×) | 1.46 (10.75 ×) |

TABLE III: Training performance comparison before and after applying PTT to previous works with CIFAR10 and DVS Gesture datasets.

| Method | Model | Dataset | Accuracy(%) (Base / PTT) | Training time(s) (Base / PTT) |
|-----------|----------|-------------|-----------------------------|----------------------------------|
| tdBN [26] | ResNet20 | CIFAR10 | 92.96 / 91.10 | 0.116 / 0.087 |
| TEBN [27] | VGG9 | CIFAR10 | 91.78 / 90.56 | 0.066 / 0.056 |
| TET [28] | VGG9 | DVS Gesture | 94.79 / 94.49 | 0.351 / 0.319 |
| NDA [29] | VGG11 | DVS Gesture | 96.88 / 95.83 | 0.299 / 0.240 |

STT. All three modules have the same number of parameters, and the differences in training time stem from architectural modifications. For the CIFAR dataset, HTT emerges as the most efficient module, reducing training latency by over 21% and FLOPs by about 8× compared to the baseline.

N-Caltech101: N-Caltech101 displays comparable trends with the CIFAR dataset across most evaluation metrics. However, the results for the HTT module differ. In contrast to the static CIFAR dataset, the accuracy of HTT is even lower than that of STT in N-Caltech101. We believe this accuracy drop is caused by the characteristics of dynamic datasets. When training an SNN architecture with static datasets, the input data remains consistent throughout the timesteps. Consequently, even with some half sub-convolutions in the later timesteps, the full sub-convolutions in the early timesteps suffice to extract feature information. However, in dynamic datasets, each input in every timestep is distinct. As a result, information loss occurs when half sub-convolutions are applied, as some information from the new input may not be effectively extracted. On the other hand, the accuracy of the PTT module surpasses that of the baseline and reduces FLOPs by about 9×. This indicates that our proposed method integrates well with not only static datasets but also dynamic event datasets.

On the Existing SNN Training Accelerator: We first directly simulate the energy costs on the existing SNN training accelerator [3] for training the baseline SNNs, the STT-based SNNs, the PTT-based SNNs, and the HTT-based SNNs. The results are shown in Fig. 4(a). Due to the model size reduction brought by the decomposition, we observe that the STT-based methods reduce 68.1% training energy cost from the baseline SNNs. However, as we discussed in Sec. IV, due to the layer-by-layer mapping strategy in the prior works, the PTT-based and HTT-based SNNs do not benefit from the parallelism during the

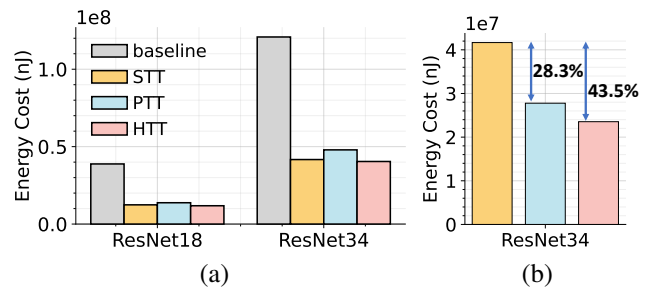


Fig. 4: (a) Training energy costs of STT, PTT, and HTT-based SNNs compared to the baseline SNN on ResNet18 and ResNet34. The results are calculated based on the accelerator design of [3]. (b) The training energy cost improvements of PTT and HTT compared to STT on our proposed multi-cluster accelerator design.

training. Consequently, compared to the training of STT-based SNNs, HTT-based SNNs cost similar energy, and the PTT-based SNNs even cost 10.9% higher energy. The reason for the higher energy cost is the fact that the PTT method needs to store the outputs from one of the sub-convolutional layers to DRAM and then re-fetch them back to the chip to merge the results before proceeding to the last sub-convolutional layer.

On the Proposed SNN Training Accelerator: Since our proposed multi-cluster design can fully harness the parallelism between sub-convolutional layers, we manage to reduce 28.3%(43.5%) training energy cost on the PTT(HTT) method from the STT method as shown in Fig. 4(b).

C. Compatibility with other SNN architectures

We further verify the compatibility of our work with previous SNN architectures, including tdBN [26], TEBN [27], TET [28], and NDA [29], by integrating the PTT modules, which is shown in Table III. We utilize the architectures given by each previous work on both static and dynamic datasets: CIFAR10 and DVS128 Gesture [32]. The PTT module can decrease the training time in all methods: 25.00% on tdBN, 15.15% on TEBN, 9.12% on TET, and 19.73% on NDA respectively, without significant accuracy degradation. These results highlight the effectiveness and flexibility of TT-SNN as a powerful plug-in tool for accelerating the training process of any SNN algorithm.

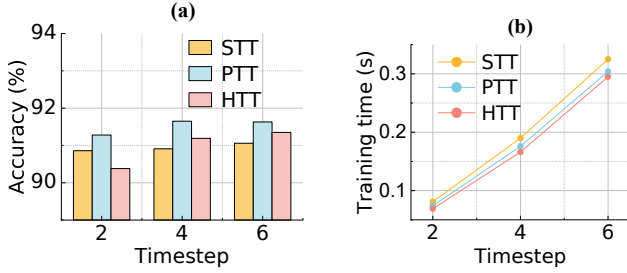


Fig. 5: Performance trends according to the timesteps. (a) Accuracy and (b) training time between STT, PTT, and HTT during the training process.

TABLE IV: Accuracy results based on the arrangement of full and half sub-convolutions in the HTT module.

| $t = 1$ | $t = 2$ | $t = 3$ | $t = 4$ | Accuracy (%) |
|---------|---------|---------|---------|--------------|
| F | F | H | H | 91.19 |
| H | H | F | F | 90.94 |
| H | F | H | F | 90.68 |
| F | H | F | H | 90.89 |

F = full sub-convolution / H = half sub-convolution

D. Ablation Study

In this section, we perform an ablation study to gain a better understanding of the TT-SNN. Specifically, we analyze the performance of TT modules based on the timestep and the placement order of full and half sub-convolutions within the HTT modules. The ablation study is conducted with the ResNet18 and CIFAR10 dataset.

TT Modules through Timestep: Given that the SNN architecture incorporates a timestep variable, it becomes crucial to examine how TT modules perform concerning this variable. Fig. 5 illustrates that our proposed TT modules work across different timesteps. Notably, the PTT module consistently achieves the highest accuracy, while the HTT module consistently exhibits the fastest training time across all timesteps.

The Order of Half Sub-convolutions: When employing the HTT module, we put the full sub-convolutions in the early timestep and the half sub-convolutions in the later timestep. This is motivated by [23], which suggests that SNN architectures tend to capture more information in the early timesteps. To further investigate this argument, we conduct experiments by altering the placement of full and half sub-convolutions within a 4-timestep ResNet18 architecture, as shown in Table IV. Note that we use two full sub-convolutions and two half sub-convolutions. As anticipated, when the full sub-convolutions are located in the early timestep, i.e., $t = 1, 2$, we achieve the highest accuracy.

VI. CONCLUSION

In this work, we have proposed TT-SNN architecture to gather several advantages of memory and computation costs during the SNN training. We first apply TT decomposition to SNN and modify the training pipeline to incorporate parallel computation instead of traditional sequential computation between TT-cores. Our extensive experiments on CIFAR10/100, and N-Caltech101 datasets validate the effectiveness of our efficient training technique for both static and dynamic datasets. Furthermore, our TT modules can be easily and flexibly

adopted in other SNN-based convolutional architectures, enabling SNNs to maintain enhanced training efficiency and reduced computational overhead in a variety of SNN applications.

ACKNOWLEDGMENTS

This work was supported in part by CoCoSys, a JUMP2.0 center sponsored by DARPA and SRC, the National Science Foundation (CA-REER Award, Grant #2312366, Grant #2318152), TII (Abu Dhabi), and the DoE MMICC center SEA-CROGS (Award #DE-SC0023198).

REFERENCES

- [1] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.
- [2] M. Davies and others, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
- [3] R. Yin *et al.*, "Sata: Sparsity-aware training accelerator for spiking neural networks," *TCAD*, 2022.
- [4] P. U. Diehl *et al.*, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International joint conference on neural networks (IJCNN)*. IEEE, 2015, pp. 1–8.
- [5] B. Han and K. Roy, "Deep spiking neural network: Energy efficiency through time based coding," in *European Conference on Computer Vision*. Springer, 2020, pp. 388–404.
- [6] Y. Wu and OTHERS, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, 2018.
- [7] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," *Advances in neural information processing systems*, vol. 31, 2018.
- [8] R. Yin *et al.*, "Workload-balanced pruning for sparse spiking neural networks," *arXiv preprint arXiv:2302.06746*, 2023.
- [9] R. V. W. Putra and M. Shafique, "Q-spinn: A framework for quantizing spiking neural networks," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.
- [10] R. Yin, Y. Li, A. Moitra, and P. Panda, "Mint: Multiplier-less integer quantization for spiking neural networks," *arXiv preprint arXiv:2305.09850*, 2023.
- [11] Q. Xu *et al.*, "Constructing deep spiking neural networks from artificial neural networks with knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 7886–7895.
- [12] R. K. Kushawaha, S. Kumar, B. Banerjee, and R. Velumrigan, "Distilling spikes: Knowledge distillation in spiking neural networks," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 4536–4543.
- [13] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.
- [14] X. Ding, Y. Guo, G. Ding, and J. Han, "Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1911–1920.
- [15] L. Liang *et al.*, "H2learn: High-efficiency learning accelerator for high-accuracy spiking neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4782–4796, 2021.
- [16] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015.
- [17] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.
- [18] X. Liu and K. K. Parhi, "Tensor decomposition for model reduction in neural networks: A review," *arXiv preprint arXiv:2304.13539*, 2023.
- [19] A. Cichocki *et al.*, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE signal processing magazine*, vol. 32, no. 2, pp. 145–163, 2015.
- [20] D. Wang, G. Zhao, G. Li, L. Deng, and Y. Wu, "Compressing 3dcnns based on tensor train decomposition," *Neural Networks*, vol. 131, pp. 215–230, 2020.
- [21] D. Lee *et al.*, "Qtnet: Quantized tensor train neural networks for 3d object and video recognition," *Neural Networks*, vol. 141, pp. 420–432, 2021.
- [22] M. Gabor and R. Zdunek, "Convolutional neural network compression via tensor-train decomposition on permuted weight tensor with automatic rank determination," in *ICCS*, 2022, pp. 654–667.
- [23] Y. Kim *et al.*, "Exploring temporal information dynamics in spiking neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 7, 2023, pp. 8308–8316.
- [24] S. Nakajima, M. Sugiyama, S. D. Babacan, and R. Tomioka, "Global analytic solution of fully-observed variational bayesian matrix factorization," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1–37, 2013.
- [25] S. Narayanan *et al.*, "Spinalflow: An architecture and dataflow tailored for spiking neural networks," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 349–362.
- [26] H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li, "Going deeper with directly-trained larger spiking neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 11 062–11 070.
- [27] C. Duan, J. Ding, S. Chen, Z. Yu, and T. Huang, "Temporal effective batch normalization in spiking neural networks," *Advances in Neural Information Processing Systems*, vol. 35, pp. 34 377–34 390, 2022.
- [28] S. Deng, Y. Li, S. Zhang, and S. Gu, "Temporal efficient training of spiking neural network via gradient re-weighting," *arXiv preprint arXiv:2202.11946*, 2022.
- [29] Y. Li *et al.*, "Neuromorphic data augmentation for training spiking neural networks," in *European Conference on Computer Vision*. Springer, 2022, pp. 631–649.
- [30] Y. Hu, L. Deng, Y. Wu, M. Yao, and G. Li, "Advancing spiking neural networks towards deep residual learning," *arXiv preprint arXiv:2112.08954*, 2021.
- [31] Y. Wu *et al.*, "Direct training for spiking neural networks: Faster, larger, better," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 1311–1318.
- [32] A. Amir *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7243–7252.