Workload-Balanced Pruning for Sparse Spiking Neural Networks

Ruokai Yin, Youngeun Kim, *Student Member, IEEE,*
Yuhang Li, Abhishek Moitra, *Student Member, IEEE,*
Nitin Satpute, Anna Hambitzer, *Member, IEEE,*
and Priyadarshini Panda, *Member, IEEE*

***Abstract*—Pruning for Spiking Neural Networks (SNNs) has emerged as a fundamental methodology for deploying deep SNNs on resource-constrained edge devices. Though the existing pruning methods can provide extremely high weight sparsity for deep SNNs, the high weight sparsity brings a workload imbalance problem. Specifically, the workload imbalance happens when a different number of non-zero weights are assigned to hardware units running in parallel, which results in low hardware utilization and thus imposes longer latency and higher energy costs. In preliminary experiments, we show that sparse SNNs (∼98% weight sparsity) can suffer as low as ∼59% utilization. To alleviate the workload imbalance problem, we propose u-Ticket, where we monitor and adjust the weight connections of the SNN during Lottery Ticket Hypothesis (LTH) based pruning, thus guaranteeing the final ticket gets optimal utilization when deployed onto the hardware. Experiments indicate that our u-Ticket can guarantee up to 100% hardware utilization, thus reducing up to 76.9% latency and 63.8% energy cost compared to the non-utilization-aware LTH method.**[1]

***Index Terms*—Spiking Neural Networks, Pruning, Neuromorphic Computing, Sparse Neural Networks**

I. Introduction

Spiking Neural Networks (SNNs) have gained tremendous attention towards ultra-low-power machine learning [1]. SNNs leverage spatio-temporal information of unary spike data to achieve energy-efficient processing in resource-constrained edge devices [2], [3]. However, in the case of large-scale tasks such as image classification, the model size of SNNs significantly increases. Unfortunately, edge devices typically have limited on-chip memory, rendering large-scale SNN deployment unpractical. To this end, recent works have proposed various unstructured SNN pruning techniques to achieve high weight sparsity in SNNs [4]–[11].

Although unstructured pruning manages to compress the SNN models into the available memory resources, sparse SNNs encounter a **workload-imbalance problem** [12]. The workload-imbalance problem comes from the conventional weight stationary dataflow [13] adopted in sparse accelerators [14]–[16]. In weight stationary dataflow, filters are divided into several groups and kept stationary inside processing elements (PEs) for filter reuse. However, different filter groups inevitably have different densities of non-zero weights due to the random weight connections from the unstructured pruning. As a result, different PEs end up with unbalanced workloads. Since all PEs run in parallel, PEs with fewer workloads must wait for the PE with the largest workload. This results in low utilization and imposes idle cycles, which increases the latency and leakage energy.

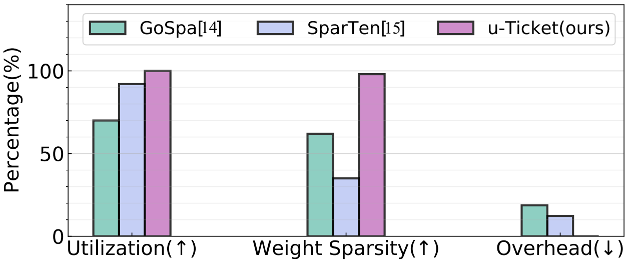


Fig. 1. Comparison between u-Ticket and state-of-the-art workload balance methods. Overall, u-Ticket recovers the PE utilization up to 100% for extremely sparse networks with 98% weight sparsity (here, we consider VGG-16). Please note that u-Ticket does not introduce any hardware area overhead, and thus is the best fit for SNNs (↑: the higher is the better, ↓: the lower is the better).

To address the workload-imbalance problem, various methods have been proposed in the prior sparse accelerator designs. However, they cannot be efficiently applied to SNNs for the following reasons. **(1) Requiring extra hardware:** The prior methods require extra hardware (*e.g.*, deep FIFOs or permuting units) [14], [15], [17]–[19] to balance the workloads. For instance, applying the hardware-based (FIFOs [14] and permuting networks [15]) workload balancing methods to SNNs require approximately 18% and 13% of extra chip area (see Fig. 1). Consequentially, the improvements in PE utilization are at the cost of additional hardware resources, which should be avoided for SNNs whose running environments are typically resource-constrained edge devices. **(2) Limited to low sparsity:** As shown in Fig. 1, the solutions from prior sparse accelerators [14], [15] only work on low sparsity (roughly 60% and 35% on VGG-16), which is not sufficient for SNNs' extremely low-power edge deployment. Moreover, the workload-imbalance problem naturally becomes more difficult to solve at high sparsity regimes. Hence, the exploration of workload balancing for extremely sparse networks ($>$ 95% weight sparsity) is missing in prior works. Considering the above-mentioned problems, we need an SNN-friendly solution to address the workload imbalance.

To this end, we propose u-Ticket, an iterative workload-balanced pruning method for SNNs that can effectively achieve

[1] Code is available at https://github.com/Intelligent-Computing-Lab-Yale/u-Ticket-Pruning

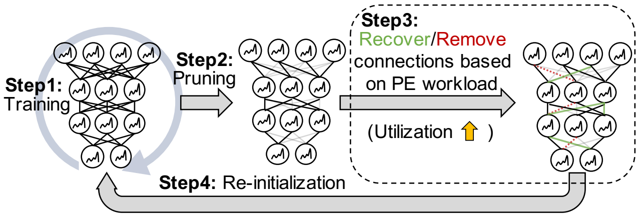


Fig. 2. Illustration of the concept of the proposed u-Ticket. Our u-Ticket consists of training (**step1**), pruning (**step2**), adjusting weight connections based on workload (**step3**), and re-initialization (**step4**). We repeat these steps for multiple rounds. Please note that the standard LTH method consists of training (**step1**), pruning (**step2**), and re-initialization (**step4**), which does not consider the utilization of the pruned SNNs.

high sparsity of weights and simultaneously minimize the workload imbalance problem. Our method is based on Lottery Ticket Hypothesis (LTH) [20], which states that sub-networks with similar accuracy can be found in over-parameterized networks by repeating *training-pruning-initialization* stages. Different from the standard LTH method [11] where the pruned networks are naively used for the next round, we either remove or recover weight connections to balance workloads across all PEs before sending the networks to re-initialization (see Fig. 2).

Compared to prior workload-balancing methods (see Fig. 1), the u-Ticket approach improves PE utilization by up to 100% (70% for [14] and 92% for [15]) while maintaining filter sparsity of 98% (60% for [14] and 35% for [15]), at iso-accuracy with the standard LTH-based pruning baseline [11]. Furthermore, since our method balances the workload during the pruning process, u-Ticket does not incur any additional hardware overhead for deployment.

We summarize the key contributions as follows:

1) We propose u-Ticket, which discovers highly sparse SNNs with optimal PE utilization. The discovered sparse SNN model achieves a similar level of accuracy, weight sparsity, and spike sparsity with the standard LTH baseline [11] while improving the utilization up to 100%.
2) By balancing the workload, u-Ticket reduces the running latency and energy cost by up to 76.9% and 63.8%, respectively, compared to the standard LTH method.
3) We extend the prior sparse accelerator [14] and propose an energy estimation model for sparse SNNs.
4) To validate the proposed u-Ticket, we conduct experiments on two representative deep architectures (i.e., VGG-16 [21] and ResNet-19 [22]) across four public datasets including CIFAR10 [23], Fashion-MNIST [24], SVHN [25], and CIFAR100 [23].

II. Related Works

A. Pruning for Spiking Neural Networks

Recently, there has been a significant growing interest in exploring spiking neural networks (SNNs) as the new generation of low-power deep neural networks under the context of edge machine learning [26], [27]. One of the main groups of SNN research focuses on the compression of the size of the networks, which is very important on edge deployment, where the memory resources are usually very limited. This work mainly focuses on one of the most popular network compression techniques: pruning.

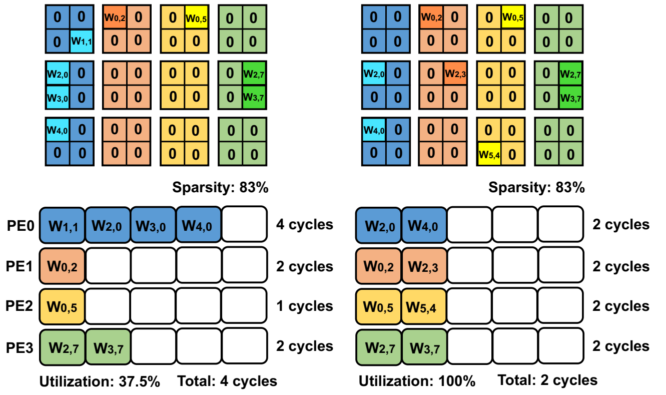


Fig. 3. Example utilization and latency resulted from imbalance and balanced workload under the same model sparsity. With the unstructured pruning, non-zero weights will have a random distribution across four groups, thus leading to unbalanced workloads across PEs as shown on the left side (PE0 has four weights assigned, while PE1 and PE2 only have one).

Pruning is a widely studied technique in the field of neural network compression that aims to reduce the size of a neural network by removing unnecessary connections or weights while maintaining its accuracy. Researchers have extensively studied the pruning for SNNs to meet the limited memory resources on the edge [4]–[11].

In [4], [5], the weight connections are removed if their magnitudes are below a pre-set threshold. In [8], the weight connections are removed according to the magnitude of the gradients. In [6], the ADMM optimization method is adopted to prune the SNNs. Both [7] and [9] propose adaptive weight pruning for SNNs. The unsupervised online weight pruning is proposed by [7] for SNNs, while [9] proposes the supervised training for the pruning masks for SNNs. A more bio-plausible pruning method for SNNs is recently proposed by [10], where the dendritic spine plasticity-based synaptic constraints are incorporated during the pruning process.

However, all those prior pruning works for SNNs are limited to relatively shallow networks ($<$ 10 layers). In [11], a lottery ticket hypothesis (LTH) based pruning method [20] is proposed for SNNs that can efficiently prune over 95% of the weight connections on very deep spiking neural networks (e.g., ResNet-19 [22]). Since the [11] shows the best performance on the deep SNN architectures (e.g., VGG-16 and ResNet-19) that many state-of-the-art SNN works adopt, we study our utilization recovery method for the LTH-based SNNs [11].

Our approach differs from other LTH-related studies. Because LTH introduces a high compression ratio with minimal performance degradation, researchers have explored its implications from various angles. Applying LTH beyond image classification is one of the major directions, such as in natural language processing [28], graph neural networks [29], and object detection [30]. Another line of work also proves the theoretical background of LTH, suggesting that the identified initial parameters might be strongly tied to the identified sparse structure [31]–[34]. The other recent work [35] studies the relationship between the LTH pruning and the LIFs' firing

TABLE I
COMPARISON WITH PRIOR WORKS ON WORKLOAD IMBALANCE PROBLEM. HW DENOTES HARDWARE.

Accelerator	**Weight Sparsity**[1]	**Extra HW**	**Utilization**[2]
EIE [17]	77%	FIFOs	∼75%
GoSPA [14]	62%	FIFOs	∼70%
SparTen [15]	35%	Shuffle Units	∼95%
Column [18]	84%	Permute Units	∼90%
u-Ticket (ours)	98%	Not Required	∼100%

[1] Measured on the VGG-like network reported in the prior works.
[2] We report the median utilization from the prior works.

probability. It further theoretically proves that the LTH holds in SNNs. Our work, different from previous research, focuses on improving the hardware efficiency (i.e., the workload imbalance problem) of the LTH-pruned SNNs.

Please note that the SNNs we focus on in this work are the ones that trained with Backpropagation through time (BPTT). This group of SNNs shows superior accuracy performance in many complex vision tasks [11], [36]–[38]. There exist other groups of SNNs that are trained differently. For example, SNNs trained with spike-timing-dependent plasticity (STDP) [4], [39], [40] and SNNs converted from pre-trained artificial neural networks [41]–[43].

B. Utilization Recovery Methods

In [14] and [17], deep FIFO queues are used to build up a backlog of workloads and thus alleviate workload imbalance passively. On the other hand, [15] and [18] address the workload imbalance problem more systematically. In [15], an offline shuffle of the weight positions is done to balance the workload in finer granularity. Similarly, [18] introduces a new training method to pack the sparse weights into a denser group for improving workload utilization. They both introduce additional hardware for permuting and unshuffling back the partial sum to the appropriate position. Although [19] proposes a utilization-aware pruning method for ANNs on speech recognition tasks, such a pruning method has not been explored in image classification. Moreover, they require the FIFO queues to achieve ∼90%utilization. Different from the prior works, our method recovers the utilization of SNNs on image classification tasks without additional hardware units for implementation. Also, we achieve significantly higher utilization (∼100%) compared to the previous works. Table I summarizes the comparison between our method and other utilization recovery methods to solve the workload imbalance problem.

III. BACKGROUND

A. Spiking Neural Networks

Spiking Neural Networks (SNNs) process the unary temporal signal through multi-layer weight connections. Instead of a ReLU neuron for a non-linear activation, recent SNN works use a Leaky-Integrate-and-Fire (LIF) neuron which contains a memory called membrane potential. The membrane potential captures the temporal spike information by storing incoming spikes and generating output spikes accordingly. Suppose a LIF neuron i has a membrane potential u_i^t at timestep t. We can formulate the discrete neuronal dynamics [44], [45] by:

$$u_i^t = \lambda u_i^{t-1} + \sum_j w_{ij} s_j^t. \tag{1}$$

Here, λ is the leaky factor for decaying the membrane potential through time. The s_j^t stands for the output spike from a neuron j at timestep t. The w_{ij} denotes a weight connection between neuron j in the previous layer and neuron i in the current layer. If the membrane potential reaches a firing threshold, the neuron generates an output spike, and the membrane potential is reset to zero. Similar to ANNs, we train the weight connection w_{ij} in all layers. Our weight optimization is based on the recently proposed surrogate gradient learning, which assumes an approximated gradient function for the non-differentiable LIF neuron [46]. We use $tanh(\cdot)$ approximation following the previous work [45].

B. Lottery Ticket Hypothesis

Lottery Ticket Hypothesis (LTH) [20] has been proposed where they found a dense neural network contains sparse sub-networks (*i.e.*, winning tickets) with similar accuracy compared to the original dense network. The winning tickets are found by multiple rounds of magnitude pruning operations. Specifically, suppose we have a dense network $f(x;\theta)$ with randomly initialized parameter weights $\theta \in \mathbb{R}^n$. In the first round, the dense network $f(x;\theta)$ is trained to convergence (**step1** in Fig. 2). Based on the trained weights, we prune $p\%$ weight connections with the lowest absolute weight values (**step2** in Fig. 2). We represent this pruning operation as a binary mask $m \in \{0,1\}^n$. In the next round, we reinitialize the pruned network with the original initialization parameters $f(x;\theta \odot m)$ (**step4** in Fig. 2), where $\odot$ represents the element-wise product. The *training-pruning-initialization* stages are repeated for multiple rounds. In the SNN domain, Kim *et al.* [11] recently applied LTH to deep SNNs, resulting in high weight sparsity (∼98%) for VGG and ResNet architectures. However, they do not consider the workload imbalance problem in sparse SNNs. Different from the previous work, we adjust weight connections for improving utilization at each pruning round **step3**, which reduces up to 77% latency and 64% energy cost compared to the standard LTH [11] while maintaining both sparsity and accuracy.

C. Workload Imbalance Problem

In the context of neural network accelerators, dataflow refers to the hardware's input and weight mapping strategy. To this effect, recent works [14]–[16], [47], [48] have demonstrated the efficacy of the weight stationary dataflow towards efficient deployment of sparse networks and SNNs.

For weight-stationary dataflow, different weights are cast to different PEs and stay inside the PE until they are maximally reused across all the relevant computations. More specifically, during the running time, depending on the memory capacity of the hardware, each layer's filter kernels will be grouped in a chosen pattern and sent to each PE. As shown in Fig.

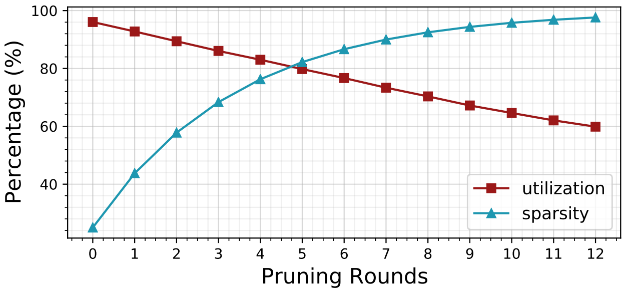


Fig. 4. Sparsity and utilization across pruning rounds for the standard LTH method without utilization awareness. The pruning is done for 13 rounds on VGG-16 being trained for image classification on CIFAR10 with 16 PEs.

3, the number of non-zero elements (or workload) allocated to each PE varies significantly due to the randomness in unstructured pruning. Moreover, the workload imbalance is persistent irrespective of the grouping method chosen. Note that here, we define the number of non-zero weights assigned to a PE as the workload.

In this case, the wasted resources in PEs are based on the difference between the largest workload and the average of all other workloads. To quantitatively measure the portion of non-wasted resources, we use the utilization metric [12], given by

$$\mu = 1 - \frac{T_{max} - T_{avg}}{T_{max}} \cdot \frac{n}{n-1}, \tag{2}$$

T_{max} and T_{avg} are the slowest and the average processing time among the PEs. n is the number of PEs. The metric quantifies the percentage of processing time that the rest of the PEs, excluding the slowest ones, are engaging in useful work.

In Fig. 4, we show how the utilization degrades as the weight sparsity of the SNN increases in the standard LTH method [11]. The preliminary result shows that in the final round, the utilization can be as low as 59% on VGG-16 CIFAR10. Here, we assume that the total number of PEs is 16, and the utilization is averaged across all layers (weighted by parameter count).

IV. u-Ticket

To resolve the workload imbalance problem, we propose u-Ticket, where we achieve high utilization in sparse SNNs during iterative pruning. In this section, we first present the algorithm to train sparse SNNs while maintaining high utilization. We then provide details of the proposed PE design and the energy model to map the u-Ticket on the hardware.

A. Algorithmic Approach

*1) **Algorithm Overview:*** Our u-Ticket pruning consists of multiple rounds similar to LTH [20]. For each round, we train the networks till convergence, prune the low-magnitude weight connections, balance the workload of PEs by recovering or removing the weight connections, and finally re-initialize the weights.

The main idea is to ensure a balanced workload between PEs after each unstructured pruning round.

The overall u-Ticket process is described in Algorithm 1. For each round, the pruned SNN from the previous round is re-initialized. After that, the model is trained and pruned where we obtain connectivity mask $\hat{m}_i$ with imbalanced PE workloads. To increase the utilization, we first compute the workload for each PE, constructing the PE workload list W^l for each layer. Based on the W^l, we calculate the average workload w^l_{avg} for layer l. Then, we go through each workload w in W^l and randomly recover $(w_{avg} - w)$ of weight connections if the PE's workload w is smaller than the average workload w^l_{avg}. Otherwise, the number of weight connections is pruned by $(w - w_{avg})$. After the workload adjustment, every workload w will have the same magnitude to ensure the optimal utilization μ. We repeat the above-mentioned stages for N rounds.

*2) **Design Choice of Workload Balance:*** There are three main design metrics to be considered for our workload balancing process: workload mapping granularity, workload checking granularity, and the workload balancing method.

Workload mapping granularity. In our u-Ticket, we assume the following procedure of mapping the weights into the PEs. For each PE, we will assign all the non-zero weights in one filter to it. Those non-zero weights will stay stationary inside the PE to fully utilize the weight-reuse across all timesteps. This weight mapping method is adopted by many recent SNN accelerator designs [47]–[49]. Moreover, similar weight-stationary mapping is also adopted in many recent sparse accelerator designs due to its dataflow efficiency under the context of sparse neural networks [14]–[16].

Workload checking granularity. In our method, we use average workload w^l_{avg} across all PEs at layer l as the reference to recover/remove weight connections. The reason behind such a design choice is as follows:

(1) If we look at only partial PE workloads to decide on a reference workload, though it will reduce the complexity of getting the average workload, it will inevitably bring a sub-optimal solution [20].

(2) The cost of checking all PE workloads to get the average workload is negligible compared to the overall iterative training-pruning-initialization process. We find that on an RTX 2080Ti GPU, the total time cost of traversing through all PEs to get the average workloads is only 0.2% of one complete LTH searching round.

Workload balancing method. In our workload balancing method, we choose to randomly recover and remove the weights to get the optimal workload. There are other criteria to choose the weights. For example, the magnitude of the weights is a very common option [50], [51].

We empirically find that randomly choosing the weights to be recovered and removed has very similar accuracy to the criteria-based choosing method. Meanwhile, the random-based choosing method has a better searching complexity with $O(n)$, while the criteria-based choosing method has at least a complexity of $O(n \cdot \log n)$.

B. Hardware Mapping

1) Processing Elements (PEs): To get an accurate energy estimation, we need to map the sparse SNN to a proper

Algorithm 1 u-Ticket

```
Input: SNNs $f(x;\theta)$ with randomly-initialized parameter weights $\theta \in \mathbb{R}^n$, connectivity mask $m_i \in \{0,1\}^n$ at iteration $i$, total pruning round $N_{Round}$, total number of layer $L$, number of PEs $n$, Workload of a PE $d$, Workload list of a layer $W^l$.
Output: Pruned $f(x;\theta_{trained} \odot m_N)$
initialize $m_1$ with 1
for $i \leftarrow 1$ to $N_{Round}$ do
    $f(x;\theta \odot m_i)$
    $f(x;\theta_{trained} \odot m_i) \leftarrow Train(f(x;\theta \odot m_i))$
    $\hat{m}_i \leftarrow Prune(f(x;\theta_{trained} \odot m_i))$
    for $l \leftarrow 1$ to $L$ do
        $W^l \leftarrow \texttt{GetWorkload}(f(x;\theta^l_{trained} \odot \hat{m}^l_i), n)$
        $d^l_{avg} \leftarrow \texttt{GetAverage}(W^l)$
        for $d$ in $W_l$ do
            if $d < d^l_{avg}$ then
                $m^l_{i+1} \leftarrow \texttt{Recover}(\hat{m}^l_i, d_{avg} - d)$
            else
                $m^l_{i+1} \leftarrow \texttt{Remove}(\hat{m}^l_i, d - d_{avg})$
            end if
        end for
    end for
end for
```

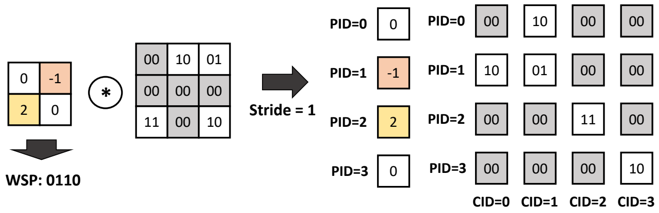


Fig. 5. Illustration of the weight sparsity pattern (WSP), the position ID (PID), and the (convolution ID) CID.

hardware design. We develop our PE design based on [14], one of the state-of-the-art sparse accelerators, to support running sparse SNNs. Please note that our method of balancing the workloads works on any sparse accelerator design as long as it utilizes the weight stationary dataflow.

First, the non-zero weights, input spikes, and their corresponding metadata (index) are read from the DRAM. The weights are represented in weight sparsity pattern (WSP) [14], while the spike activations are represented in standard compressed sparse row (CSR) format. We use four timesteps for the SNN in our experiments, thus we can group every two activations into one byte (each activation has four unary spikes.)

Then, an activation processing unit (APU, outside PEs) filters out the zero activation (0-spikes across four timesteps) and sends the non-zero activation together with their position indices (decoded from CSR) to the PE arrays. The position indices help to match the non-zero weights and activation in 2-D convolution.

In Fig. 5, we further illustrate the position indices that we used in this work. We use PID (Position ID) and CID (Convolution ID) to match the valid combination of spike activations and weights (both non-zero). We explain the assignment of PID and CID on the right of Fig.5. For an unrolled 2-D convolution map, all the activations involved with the same inner product (in the same column) share the same CID. And the activations on the same row share a PID, as they correspond to the same weight which is also assigned with the same PID value. With this CID/PID matching, it is very convenient to match the non-zero pairs of weights and spike activations [14].

At the PE level, each PE contains four 16-bit AND gates, 256 24-bit accumulators, and one 1024 × 16 bits SRAM-based scratch-pad. We further extend the 256 accumulators with 256 LIF units for generating the output spikes. Each LIF unit is equipped with four 24-bit registers for storing the membrane potential across four timesteps.

Fig. 6 illustrates the overall architecture and the computation flow inside the PE. We process the network in a tick-batched manner [47]. At step ❶, the non-zero weights and their WSPs are mapped to each PE. At step ❷, the spike activation S_{in} together with their position indices are sent to PE. Based on the weight's WSP and the activation's position index, the selector unit will output the matched non-zero weight. At step ❸ and ❹, the dot-product operations between the input spike and the matched weights are carried out, and the partial sums are stored according to their position index. At step ❺, the partial sums for each time step are sequentially sent to the LIF units to generate the output spikes for each time step. Note that steps ❸ - ❺ need to be repeated four times to match the four timesteps used in our SNN model (only 1 bit of S_{in} is cast to the PE at a time in step ❷).

2) Energy Modeling: We do the simulation for the full architecture. Since u-Ticket balances the workloads between PEs, most of the improvements can be found at the PE level. Thus, this work focuses on energy estimation at the PE level. We extend the energy model from [48] to estimate the total energy:

$$E_{total} = N_{work} \cdot (E^d_{PE} \cdot (1 - S^{spa}_{in}) + E^l_{PE}) + N_{idle} \cdot E^l_{PE}, \quad (3)$$

where E^d_{PE} and E^l_{PE} are the dynamic and leakage energy of a single PE processing one input spike. As shown in [48], there is no extra cost for skipping the zero-spike computation in SNNs. In this work, we directly apply the input spikes as the enable signal of the accumulators and LIF units. In this way, we can stop those circuits from flipping when there are incoming zero spikes. Thus, we directly apply the term of spike sparsity, S^{spa}_{in}, in Eqn. 3 to approximate the dynamic energy saving by skipping the zero spikes. Here N_{work} is defined as the total work cycles in which PEs are doing useful work and N_{idle} denotes the total cycles in which PEs are waiting in an idle state.

V. EXPERIMENT

A. Experimental Settings

1) Software Configuration: First, to validate the u-Ticket pruning method, we evaluate our u-Ticket methods on four public datasets: CIFAR10 [23], Fashion-MNIST [24], SVHN [25], and CIFAR100 [23]. We choose two representative deep network architectures: VGG-16 [21] and ResNet-19 [22]. We implement the networks on PyTorch and set the

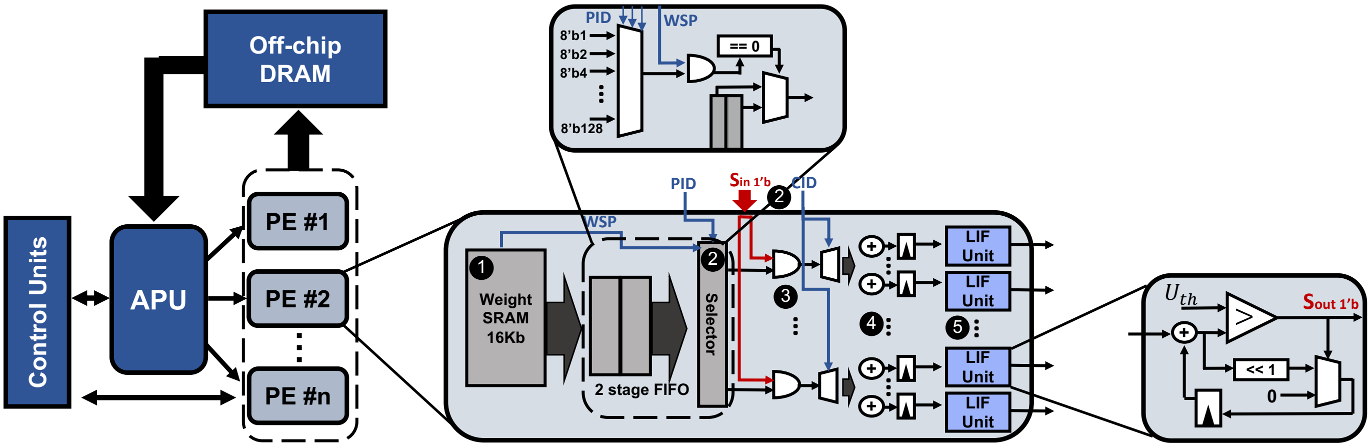


Fig. 6. Overall architecture and the detailed inner architecture of PE. Here APU denotes the activation processing unit.

TABLE II
SNN TRAINING HYPERPARAMETERS FOR OUR U-TICKET METHOD.

Parameters	Description	Quantity
Batch Size	-	128
Optimizer	-	SGD
T	timesteps	4
γ	learning rate	1e-1
λ	weight decay	5e-4
μ	momentum	0.9
τ	membrane potential leak	0.75
v_{th}	firing threshold	1
reset mode	-	hard
epoch	number of training epochs	150

timesteps T to 4 for all experiments. We use the state-of-the-art direct encoding technique that has been shown to train SNNs on image classification datasets with very few timesteps. We use the same training configurations used in Table. II.

2) Hardware Configuration: We report the utilization, latency, work cycles, and idle cycles based on our PyTorch-based simulator which simulates the running-time distribution of the weights to PEs. We use the weights grouping method as in [14], [48] with 16 PEs. The PE level energy is estimated with the model in Section IV-B2 with all computing units synthesized in Synopsys Design Compiler at 400MHz using 32nm CMOS technology and the memory units simulated in CACTI. We set the standard LTH method [11] without utilization-awareness as our baseline and use the same estimation model to get the speed-up and energy results.

B. Experimental Results

1) Validation Result: We summarize the validation results in Table III. The results confirm that our method works well for deep SNNs (less than ∼1% accuracy drop). We also compare the sparsity of filters and spikes between these two methods. u-Ticket has a slightly higher filter sparsity, due to the extra reduction in weight connections to ensure balanced workloads for each PE. At the same time, u-Ticket keeps a similar level of spike sparsity on VGG-16 and has better spike sparsity on ResNet-19. While a higher spike sparsity will bring better energy efficiency, a spike sparsity that is too high will cause an accuracy drop in deep SNNs [52]. This explains the accuracy-sparsity tradeoff on ResNet-19 (on average 0.76% accuracy drop with 3.5% sparsity gain).

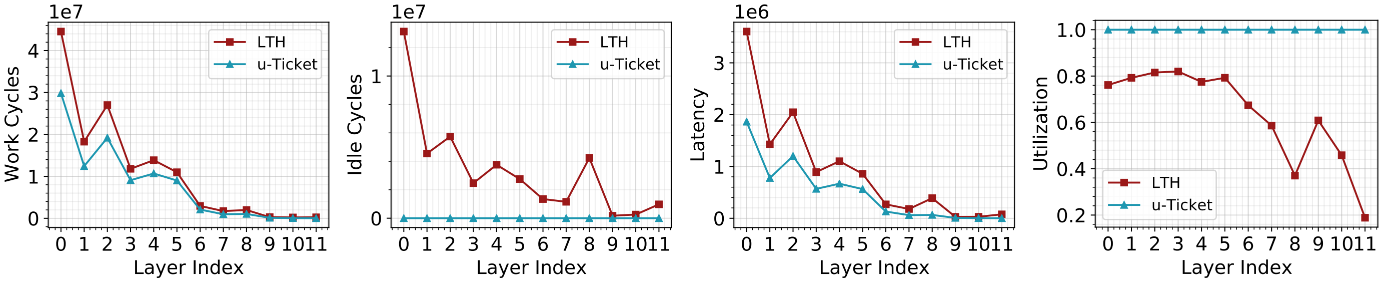
TABLE III
COMPARISON OF ACCURACY, SPARSITY OF FILTERS, AND SPARSITY OF SPIKES BETWEEN OUR METHOD AND THE STANDARD LTH METHOD.

Dataset	Method	Acc.(%)	Sparsity(%) (filters)	Sparsity(%) (spikes)
		VGG-16 [21]		
CIFAR10	LTH [11]	**91.0**	98.2	84.8
	u-Ticket (ours)	90.7	**98.4**	**85.9**
FMNIST	LTH [11]	**94.6**	98.2	**83.9**
	u-Ticket (ours)	94.0	**98.5**	81.4
SVHN	LTH [11]	**95.5**	98.2	**84.9**
	u-Ticket (ours)	94.8	**98.5**	80.1
CIFAR100	LTH [11]	**63.9**	98.2	81.9
	u-Ticket (ours)	63.1	**98.2**	**82.0**
		ResNet-19 [22]		
CIFAR10	LTH [11]	**91.0**	97.6	64.1
	u-Ticket (ours)	90.3	**98.4**	**68.3**
FMNIST	LTH [11]	**94.4**	98.2	60.1
	u-Ticket (ours)	93.3	**99.0**	**62.9**
SVHN	LTH [11]	**95.1**	97.6	63.6
	u-Ticket (ours)	94.6	**98.6**	**68.2**
CIFAR100	LTH [11]	**66.7**	**98.2**	77.5
	u-Ticket (ours)	66.3	98.1	**77.6**

In addition, we show the convergence speed results between our method and the original LTH method in Fig. 8. The result shows that on the CIFAR-10 dataset with the VGG-16 network, our method only brings a slight convergence overhead in the first 50 epochs. After the 100 epoch, the convergence difference between the two methods is already negligible. The same trends can also be found on other datasets and networks.

2) Hardware Performance: We consider four metrics in this section (*i.e.*, work cycles, idle cycles, latency, and utilization).

- **Work cycles** (N_{work} in Eqn. 4): Sum of total work cycles for every PE across all the layers in the network.



(a) (b) (c) (d)

Fig. 7. The layerwise performance comparison between LTH and u-Ticket on four metrics, *i.e.*, (a) work cycles, (b) idle cycles, (c) latency, (d) utilization. We conduct experiments with VGG-16 architecture on CIFAR10.

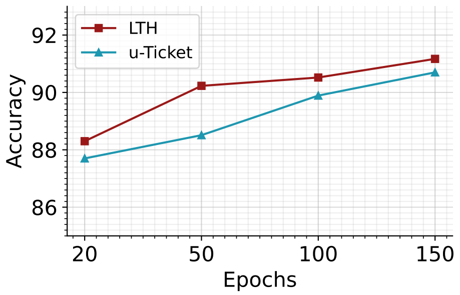


Fig. 8. Comparison of the convergence speed between the original LTH method and our method. The accuracy results are based on CIFAR-10 with VGG-16 networks.

- **Idle cycles** (N_{idle} in Eqn. 4): Sum of total idle cycles for every PE across all the layers in the network.
- **Latency**: Time required by PEs to process all the layers in the network. The latency is normalized with respect to the time required for a PE to process one input spike.
- **Utilization**: We use Eqn. 2 to compute the utilization for each layer. To compute the utilization of the network, we calculate the weighted average utilization.

The hardware improvement results are summarized in Table IV. By iteratively applying the utilization recovery during the pruning, u-Ticket can recover the utilization up to 100% in the final pruning round, thus reducing almost all the idle cycles for PEs. Because of the re-balance of workloads among PEs, the network can leverage more parallelism from the PE array, thus significantly reducing the running latency. The number of work cycles stays similar on both networks. We further visualize the layerwise speedup results for VGG-16 on CIFAR10 in Fig. 7. Overall, the layerwise work cycles and latency share similar trends between the two methods. Furthermore, u-Ticket has a larger number of idle cycle reductions on earlier layers due to the larger feature map sizes.

3) Energy Performance: In this section, we further show the energy efficiency improvements of u-Ticket over the standard LTH baseline. The energy differences are visualized in Fig. 9 (a), from which we observe that the energy benefits of balancing the workloads are huge. For CIFAR10, FMNIST, and SVHN, we managed to reduce the energy cost by 41.8%, 35.4%, and 37.2% on VGG-16, and 55.5%, 63.8%, and 56.1% on ResNet-19.

TABLE IV
COMPARISON OF WORK CYCLES, IDLE CYCLES, LATENCY, AND UTILIZATION BETWEEN U-TICKET AND THE STANDARD LTH.

Dataset	Method	Work ($\times 1e8$)	Idle ($\times 1e8$)	Latency ($\times 1e8$)	Utilization
		VGG-16 [21]			
CIFAR10	LTH [11]	1.34	0.41	0.11	0.59
	u-Ticket (ours)	**0.94**	**0.00**	**0.06**	**1.00**
FMNIST	LTH [11]	1.10	0.42	0.10	0.57
	u-Ticket (ours)	**0.81**	**0.00**	**0.05**	**1.00**
SVHN	LTH [11]	1.15	0.69	0.12	0.47
	u-Ticket (ours)	**0.86**	**0.00**	**0.05**	**1.00**
CIFAR100	LTH [11]	1.15	0.69	0.12	0.47
	u-Ticket (ours)	**0.86**	**0.00**	**0.05**	**1.00**
		ResNet-19 [22]			
CIFAR10	LTH [11]	1.66	1.73	0.21	0.31
	u-Ticket (ours)	**1.10**	**0.00**	**0.07**	**1.00**
FMNIST	LTH [11]	1.26	1.89	0.20	0.27
	u-Ticket (ours)	**0.73**	**0.00**	**0.05**	**1.00**
SVHN	LTH [11]	1.27	1.34	0.16	0.30
	u-Ticket (ours)	**0.84**	**0.00**	**0.05**	**1.00**
CIFAR100	LTH [11]	1.15	0.69	0.12	0.47
	u-Ticket (ours)	**0.86**	**0.00**	**0.05**	**1.00**

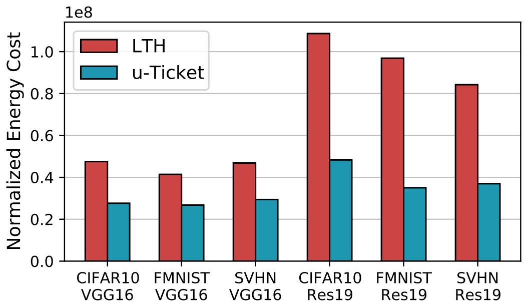


Fig. 9. Comparison of the normalized energy cost between two networks and across three datasets. The energy results are normalized to the energy required by a PE to process one input spike.

The main source of energy cost reduction comes from the elimination of idle cycles and the reduction of latency, which ultimately reduces the leakage energy of the hardware. ResNet-19, whose network is deeper, suffers more from the workload imbalance problem and thus has more idle cycles and longer latency compared to VGG-16. By eliminating

TABLE V
COMPARISON BETWEEN THE PERFORMANCE OF U-TICKET WITH AND WITHOUT THE EARLY-TICKET (ET) METHOD.

Method	Acc.(%)	Sparsity(%) (filters)	Utilization	Searching Time (seconds/round)
		VGG-16, CIFAR10		
LTH [11]	**91.0**	98.2	0.66	3031
u-Ticket	90.7	**98.4**	**1.00**	3032
LTH ET [11]	90.9	98.2	0.59	1553
u-Ticket ET	90.6	**98.3**	**1.00**	1558

almost all the idle cycles, u-Ticket brings more energy cost reduction to ResNet-19 compared to VGG-16.

4) Searching and Recovering Speed: There are several techniques in other LTH-based work that are used to reduce the searching time [11], [53]. We further apply the Early-Ticket [11] (ET) to u-Ticket to reduce the ticket searching time for SNNs. As shown in the Table. V, u-Ticket works well with ET. By applying ET, u-Ticket is still able to recover the utilization to 100% at iso-accuracy and weight sparsity, with approximately 50% searching time reduction. The result suggests that our proposed method is orthogonal to other existing techniques that reduce the searching time for standard LTH.

C. Ablation Studies

Analysis of Sparsity:

We study the effects of the u-Ticket method under different weight sparsity. We measure the energy difference between u-Ticket and the LTH baseline at different pruning rounds for both ResNet-19 and VGG-16 on the CIFAR10 dataset. The result is visualized in Fig. 10 (b). As observed, with increased weight sparsity, the benefits of using u-Ticket get larger. This is due to the degradation of the utilization in LTH as aforementioned in Fig. 4.

Analysis of #PEs:

We further study the effects of changing the number of PEs. We run the u-Ticket for VGG-16 on CIFAR10 with 2, 4, 8, 16, 32, and 64 PEs, and illustrate the results in Fig. 10 (a). While the energy cost only slightly changes with the increasing number of PEs, the latency decreases linearly. Considering that the area of PE arrays will also linearly increase with the number of PEs, we conduct most of our experiments with 16 PEs, which is a suitable trade-off point.

Analysis of Energy Breakdown:

In Fig. 11, we show the energy breakdown comparison between u-Ticket and the LTH baseline on ResNet-19 for the CIFAR10 dataset. The energy components are the dynamic and leakage energy of MAC operation, LIF operation, and MEM operation (reading of SRAM-based scratchpad). We observe that the leakage energy for both MAC and LIF operation is significantly reduced in u-Ticket due to the elimination of the idle cycles. Expectedly, the portion of the dynamic energy of MAC and LIF operation increases.

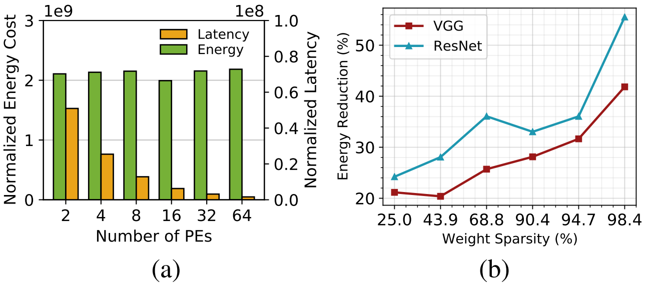


Fig. 10. (a) Comparison of the normalized energy cost between two networks and across three datasets. The energy results are normalized to the energy required by a PE to process one input spike. (b) Percentage of normalized energy reduction compared to the LTH baseline for different weight sparsity.

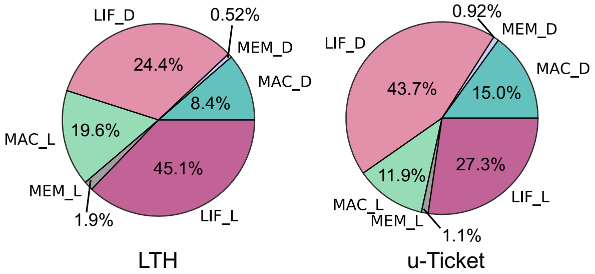


Fig. 11. Comparison of the energy breakdown between u-Ticket and the LTH baseline. MAC_L, MEM_L, and LIF_L denote the leakage energy for MAC, LIF, and memory operation, while MAC_D, MEM_D, and LIF_D denote their dynamic energy.

System Level Study: Furthermore, we study the behavior of the overall system of sparse SNNs. In Fig. 12 (a), we show how the total DRAM and SRAM access (normalized with respect to dense SNN) decrease with increasing weight sparsity. The results again encourage the necessity of pruning the networks into the extremely high sparsity domain. Furthermore, we find that in the extremely high weight sparsity regime, the PE level energy starts to take a significant portion of the total energy ($\sim$ 45% on VGG-16 with CIFAR10). As a result, after applying u-Ticket to balance the PE workloads, we managed to reduce approximately 19% of the total energy at the system level as shown in Fig. 12 (b).

u-Ticket Utilization Recovery Overheads on GPUs: We also quantify the latency overheads of utilization check and recovery on multiple GPU devices. The result shows that our u-Ticket brings almost no latency overheads to the standard LTH across three CUDA GPUs: RTX-2080Ti, V100, and A100. The result is shown in Fig. 13. The latency is the total searching time of one complete u-Ticket search round. The green portion is the time for training on VGG-16 for the CIFAR10 dataset for 15 epochs. The orange part is the utilization recovery and check part (Line 6-16 in Algorithm 1). Compared to the training time, u-Ticket's utilization check and recovery time is negligible ($\sim$ 0.3% of one complete ticket searching time).

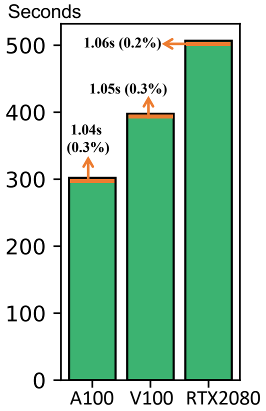
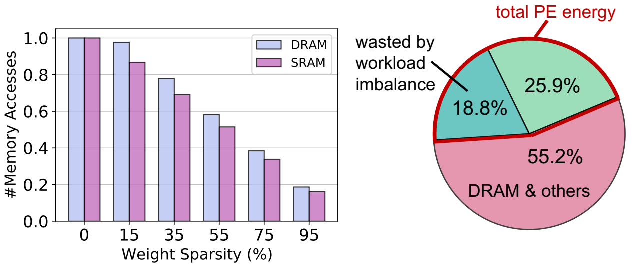


Fig. 13. GPU latency of u-Ticket.



(a) (b)

Fig. 12. (a) Normalized DRAM and SRAM accesses comparison across different weight sparsity. (b) The component breakup of the total energy for LTH baseline with 95% sparsity. Both results are shown for VGG-16 with CIFAR10.

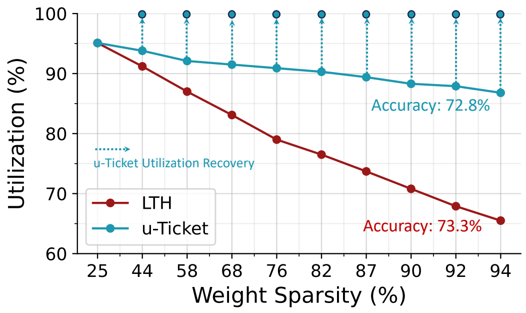


Fig. 14. Weight sparsity vs. utilization for CIFAR10-DVS datasets on ResNet-19 model.

TABLE VI
ABLATION STUDY: COMPARISON OF ACCURACY, SPARSITY OF FILTERS, AND UTILIZATION BETWEEN U-TICKET AND LTH METHOD ON DVS DATASETS.

Dataset	Method	Acc.(%)	Sparsity(%) (filters)	Utilization
		ResNet-19 [22]		
CIFAR10-DVS	LTH [11]	**73.3**	94.4	0.66
	u-Ticket (ours)	72.8	**95.3**	**1.00**
N-Caltech101	LTH [11]	63.2	**98.2**	0.54
	u-Ticket (ours)	**64.9**	97.8	**1.00**

Evaluation on Temporal-Series Datasets:

To further validate our method on datasets that heavily rely on temporal-series information, we conduct experiments using Dynamic Vision Sensor (DVS) datasets obtained from event-based cameras. Descriptions of these datasets are provided below:

CIFAR10-DVS: CIFAR10-DVS dataset [54] contains 10K DVS images recorded from the CIFAR10 dataset [23]. We resize the image resolution to 48 x 48 and divide the event series into 10 frames per image sample.

N-Caltech 101: N-Caltech 101 dataset [55] contains 8831 DVS images recorded from the Caltech 101 dataset. Similar to the CIFAR10-DVS, we resize the image resolution to 48 x 48 and divide the event data into 10 frames per sample. We run both datasets on the ResNet-19 network with 10 rounds of u-Ticket search. At each round, 25% weights are pruned.

We validate our u-Ticket method in Table. VI. For both the DVS datasets, our method achieves iso-accuracy compared to the original LTH method with slightly higher weight sparsity and 100% utilization. We further illustrate the trend of utilization and weight sparsity on CIFAR10-DVS in Fig. 14. Our method yields a ticket with better utilization in every round. Thus, we can easily recover the utilization to 100% for that ticket without accuracy degradation.

VI. Discussion

A. Comparison with Structured Pruning

In this work, we target to solve the workload imbalance problem associated with unstructured pruning. In contrast to unstructured pruning, structured pruning has also been a very popular network compression method [50], [56], [57]. In structured pruning, the networks are pruned in a pattern that aims to leverage the hardware's power to process the pruned networks more efficiently. The nature of the structured pruning does not make it suffer from the workload imbalance problem that we have discussed in this work. Although the structured pruning methods take advantage of efficient hardware processing, they do suffer from a relatively lower weight sparsity. For example, on the VGG-16 network, the structured pruning on average achieves around 85% weight sparsity, while our LTH-based unstructured pruning gets over 95% weight sparsity.

B. Compatibility on Async Neuromorphic Chip

While in the paper, we have limited our discussion of the uTicket's hardware benefits to the synchronized digital accelerators. It is worth noting that our method also has the potential to improve the utilization of the async neuromorphic chips [2], [3], [58] when deploying the LTH-based SNN models. Assume that a sparse SNN is deployed on the neuromorphic chip. Depending on the number of synaptic connections, different post-synaptic neurons will receive different numbers of event-driven packages between timestep t to $t + 1$. This will result in an imbalanced processing time across different cores. Furthermore, as a popular design choice in neuromorphic chips, every time the chip advances its timestep, there will be a barrier synchronization [59] between each core. Consequently, the heavily imbalanced sparse networks will lead to a longer waiting time for the idling cores during the barrier synchronization process. So, in conclusion, as long as the neuromorphic chips take some synchronization steps between the asynchronized computations, there will be a workload imbalance problem. Our method can potentially provide a workload balance solution without any hardware modifications on the chip. Moreover, this discussion should also apply to other chip designs that use addressing algorithms.

C. Future Direction

We suggest several interesting potential future directions based on this work. Firstly, although we find that the u-Ticket method works well on recovering the utilization at iso-accuracy and iso-weight-sparsity, the observations are based on empirical experiment results. A detailed analysis of the mathematical reason behind this would be useful for the community. Moreover, although we have the experiment results showing that our workload balancing method would not hurt the sparse firing activity of SNNs, theoretically studying the relationship between the workload utilization and the spike firing activity of SNNs is important. We envision that the probability model from [35], which builds the relationship between pruning and SNN firing, would be a good starting point. Further, the motivation of this work is based on the hardware-resource-limitation of SNNs, thus we focus our experiments and analysis on SNNs. However, whether our method is applicable to ANNs can be another useful insight for the community.

VII. Conclusion

In this work, we propose u-Ticket, a utilization-aware LTH-based pruning method that solves the workload imbalance problem in SNNs. Unlike prior works, u-Ticket recovers the utilization during pruning, thus avoiding additional hardware to balance the workloads during deployment. Additionally, at iso-accuracy, u-Ticket improves PE utilization by up to 100% compared to the standard LTH-based pruning method while maintaining filter sparsity of 98%. Moreover, u-Ticket reduces the running latency by up to 77% and energy cost by up to 64% compared to the standard LTH baseline.

VIII. Acknowledgements

We thank our reviewers for their valuable feedback. This work was supported in part by CoCoSys, a JUMP2.0 center sponsored by DARPA and SRC, the National Science Foundation (CAREER Award, Grant #2312366, Grant #2318152), TII (Abu Dhabi), and the DoE MMICC center SEA-CROGS (Award #DE-SC0023198).

References

[1] K. Roy *et al.*, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[2] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[3] F. Akopyan *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE TCAD*, vol. 34, no. 10, pp. 1537–1557, 2015.

[4] N. Rathi *et al.*, "Stdp-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 668–677, 2018.

[5] E. O. Neftci *et al.*, "Stochastic synapses enable efficient brain-inspired learning machines," *Frontiers in neuroscience*, vol. 10, p. 241, 2016.

[6] L. Deng *et al.*, "Comprehensive snn compression using admm optimization and activity regularization," *IEEE TNNLS*, 2021.

[7] W. Guo *et al.*, "Unsupervised adaptive weight pruning for energy-efficient neuromorphic systems," *Frontiers in Neuroscience*, vol. 14, p. 598876, 2020.

[8] Y. Chen *et al.*, "Pruning of deep spiking neural networks through gradient rewiring," *arXiv preprint arXiv:2105.04916*, 2021.

[9] Y. Shi *et al.*, "A soft-pruning method applied during training of spiking neural networks for in-memory computing applications," *Frontiers in neuroscience*, vol. 13, p. 405, 2019.

[10] B. Han *et al.*, "Adaptive sparse structure development with pruning and regeneration for spiking neural networks," *arXiv preprint arXiv:2211.12219*, 2022.

[11] Y. Kim *et al.*, "Exploring lottery ticket hypothesis in spiking neural networks," in *European Conference on Computer Vision*. Springer, 2022, pp. 102–120.

[12] L. DeRose *et al.*, "Detecting application load imbalance on high end massively parallel systems," in *EuroPar*, 2007.

[13] Y.-H. Chen *et al.*, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ISCA*, 2016.

[14] C. Deng *et al.*, "Gospa: an energy-efficient high-performance globally optimized sparse convolutional neural network accelerator," in *ISCA*, 2021.

[15] A. Gondimalla *et al.*, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *MIRCRO*, 2019.

[16] A. Parashar *et al.*, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ISCA*, 2017.

[17] S. Han *et al.*, "Eie: Efficient inference engine on compressed deep neural network," *ISCA*, 2016.

[18] H. Kung *et al.*, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *ASPLOS*, 2019.

[19] S. Han *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *FPGA*, 2017.

[20] J. Frankle *et al.*, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv:1803.03635*, 2018.

[21] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[22] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016.

[23] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," 2009.

[24] H. Xiao *et al.*, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv:1708.07747*, 2017.

[25] Y. Netzer *et al.*, "Reading digits in natural images with unsupervised feature learning," 2011.

[26] K. Roy *et al.*, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[27] D. V. Christensen *et al.*, "2022 roadmap on neuromorphic computing and engineering," *Neuromorphic Computing and Engineering*, vol. 2, no. 2, p. 022501, 2022.

[28] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, and M. Carbin, "The lottery ticket hypothesis for pre-trained bert networks," *Advances in neural information processing systems*, vol. 33, pp. 15 834–15 846, 2020.

[29] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, "A unified lottery ticket hypothesis for graph neural networks," in *International conference on machine learning*. PMLR, 2021, pp. 1695–1706.

[30] S. Girish, S. R. Maiya, K. Gupta, H. Chen, L. S. Davis, and A. Shrivastava, "The lottery ticket hypothesis for object recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 762–771.

[31] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," in *International Conference on Machine Learning*. PMLR, 2020, pp. 6682–6691.

[32] A. Pensia, S. Rajput, A. Nagle, H. Vishwakarma, and D. Papailiopoulos, "Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient," *Advances in neural information processing systems*, vol. 33, pp. 2599–2610, 2020.

[33] R. Burkholz, "Most activation functions can win the lottery without excessive depth," *Advances in Neural Information Processing Systems*, vol. 35, pp. 18 707–18 720, 2022.

[34] A. da Cunha, E. Natale, and L. Viennot, "Proving the lottery ticket hypothesis for convolutional neural networks," in *International Conference on Learning Representations*, 2021.

[35] M. Yao, Y. Chou, G. Zhao, X. Zheng, Y. Tian, B. Xu, and G. Li, "Probabilistic modeling: Proving the lottery ticket hypothesis in spiking neural network," *arXiv preprint arXiv:2305.12148*, 2023.

[36] C. Lee *et al.*, "Enabling spike-based backpropagation for training deep neural network architectures," *Frontiers in neuroscience*, 2020.

[37] Y. Wu *et al.*, "Direct training for spiking neural networks: Faster, larger, better," in *AAAI*, 2019.

[38] H. Zheng *et al.*, "Going deeper with directly-trained larger spiking neural networks," in *AAAI*, 2021.

[39] P. U. Diehl *et al.*, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, p. 99, 2015.

[40] R. Vaila, J. Chiasson, and V. Saxena, "A deep unsupervised feature learning spiking neural network with binarized classification layers for the emnist classification," *IEEE transactions on emerging topics in computational intelligence*, vol. 6, no. 1, pp. 124–135, 2020.

[41] N. Rathi *et al.*, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," *arXiv preprint arXiv:2005.01807*, 2020.

[42] S. Deng *et al.*, "Optimal conversion of conventional artificial neural networks to spiking neural networks," *arXiv preprint arXiv:2103.00476*, 2021.

[43] A. Sengupta *et al.*, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in neuroscience*, vol. 13, p. 95, 2019.

[44] Y. Wu *et al.*, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, p. 331, 2018.

[45] W. Fang *et al.*, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *ICCV*, 2021.

[46] E. O. Neftci *et al.*, "Surrogate gradient learning in spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, pp. 61–63, 2019.

[47] S. Narayanan *et al.*, "Spinalflow: An architecture and dataflow tailored for spiking neural networks," in *ISCA*, 2020.

[48] R. Yin *et al.*, "Sata: Sparsity-aware training accelerator for spiking neural networks," *arXiv:2204.05422*, 2022.

[49] J.-J. Lee *et al.*, "Parallel time batching: Systolic-array acceleration of sparse spiking neural computation," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 317–330.

[50] W. Wen *et al.*, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[51] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[52] Y. Li *et al.*, "Differentiable spike: Rethinking gradient-descent for training spiking neural networks," *NeurIPS*, 2021.

[53] H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, R. G. Baraniuk, Z. Wang, and Y. Lin, "Drawing early-bird tickets: Towards more efficient training of deep networks," *arXiv preprint arXiv:1909.11957*, 2019.

[54] H. Li *et al.*, "Cifar10-dvs: an event-stream dataset for object classification," *Frontiers in neuroscience*, vol. 11, p. 309, 2017.

[55] G. Orchard *et al.*, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015.

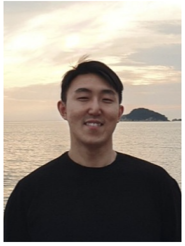
[56] E. Hanson *et al.*, "Cascading structured pruning: enabling high data reuse for sparse dnn accelerators," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 522–535.

[57] A. Zhou *et al.*, "Learning n: M fine-grained structured sparse neural networks from scratch," *arXiv preprint arXiv:2102.04010*, 2021.

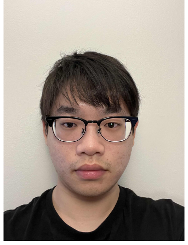
[58] O. J. Richter *et al.*, "Event-driven spiking convolutional neural network," Jun. 16 2022, uS Patent App. 17/601,939.

[59] D. Hensgen *et al.*, "Two algorithms for barrier synchronization," *International Journal of Parallel Programming*, vol. 17, pp. 1–17, 1988.

Ruokai Yin is a Ph.D. student in the Department of Electrical Engineering at Yale University, advised by Prof. Priyadarshini Panda. His research interests lie in designing high-performance computer architectures for neural networks. Prior to joining Yale, he received his BS-Electrical Engineering degree from the University of Wisconsin-Madison, where he worked with Prof. Joshua San Miguel on computer architectures for stochastic computing.



Youngeun Kim is currently working toward a Ph.D. degree in Electrical Engineering at Yale University, New Haven, CT, USA. Prior to joining Yale, he worked as a full-time student intern at T-Brain, AI Center, SK telecom, South Korea. He received his B.S. degree in Electronic Engineering from Sogang University, South Korea, in 2018 and M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology (KAIST), in 2020. His research interests include neuromorphic computing, computer vision, and deep learning.



Yuhang Li received the B.E. in Department of Computer Science and Technology, University of Electronic Science and Technology of China (UESTC) in 2020. He was a research assistant at the National University of Singapore and UESTC in 2019 and 2021, respectively. Now he is pursuing his Ph.D. degree at Yale University, supervised by Prof. Priyadarshini Panda. His research interests include Efficient Deep Learning, Brain-inspired Computing, and Model Compression.

Abhishek Moitra is pursuing his Ph.D. in the Intelligent Computing Lab at Yale. His research works have been published in reputed journals such as IEEE TCAS-1, IEEE TCAD and conferences such as DAC. His research interests involve hardware-algorithm co-design and co-exploration for designing robust and energy-efficient hardware architectures for deep learning tasks.

Priyadarshini Panda is an assistant professor in the electrical engineering department at Yale University, USA. She received her B.E. degree in Electrical & Electronics and Master's degree in Physics from BITS, Pilani, India in 2013 and her Ph.D. in Electrical & Computer Engineering from Purdue University, USA in 2019. She was the recipient of outstanding student award in Physics in 2013. In 2017, she interned at Intel Labs, Oregon, USA where she developed large-scale spiking neural network algorithms for benchmarking the Loihi chip. She is the recipient of the 2019 Amazon Research Award, 2022 Google Scholar Research Award, and 2022 DARPA Riser Award. She has published more than 60 publications in well-recognized venues including, Nature, Nature Communications, and IEEE among others. Her research interests include-neuromorphic computing, energy-efficient deep learning, adversarial robustness, and hardware-centric design of robust neural systems.