MINT: Multiplier-less INTeger Quantization for Energy Efficient Spiking Neural Networks

Ruokai Yin, Yuhang Li, Abhishek Moitra and Priyadarshini Panda
Department of Electrical Engineering, Yale University. USA
Email: {ruokai.yin, yuhang.li, abhishek.moitra, priya.panda}@yale.edu

***Abstract*—We propose Multiplier-less INTeger (MINT) quantization, an efficient uniform quantization scheme for the weights and membrane potentials in spiking neural networks (SNNs). Unlike prior SNN quantization works, MINT quantizes the memory-hungry membrane potentials to extremely low precision (2-bit) to significantly reduce the total memory footprint. Additionally, MINT quantization shares the quantization scaling factor between the weights and membrane potentials, eliminating the need for multipliers that are necessary for vanilla uniform quantization. Experimental results demonstrate that our proposed method achieves accuracy that matches the full-precision models and other state-of-the-art SNN quantization works while outperforming them on total memory footprint and hardware cost at deployment. For instance, 2-bit MINT VGG-16 achieves 90.6% accuracy on CIFAR-10 with approximately 93.8% reduction in total memory footprint from the full-precision model; meanwhile, it reduces 90% computation energy compared to the vanilla uniform quantization at deployment.**

***Index Terms*—Spiking neural networks, Quantization, Neuromorphic computing, Computer architecture**

I. Introduction

Spiking Neural Networks (SNNs) [1] are a promising alternative to Artificial Neural Networks (ANNs). They stand out for their energy efficiency, which is attained by handling extremely sparse unary spike trains (0,1) over discrete timesteps. This makes SNNs appealing for low-power edge devices, as they can run with greatly simplified arithmetic units. Recent backpropagation-through-time (BPTT)-based SNN efforts [2]–[4] have attained accuracy levels in complicated vision tasks [5] that are on par with ANNs. However, the curse of dimensionality has also affected SNNs where model sizes have exploded to achieve competitive accuracy on complex tasks. As a result, SNNs become incompatible with edge devices that have memory constraints. Recent efforts have suggested compressing BPTT-based SNNs by quantization to overcome this [6]–[8]. Although the earlier research was successful in reducing the memory size of the weights on BPTT-based SNNs, two critical issues have been overlooked.

Firstly, the size of membrane potential has not received enough attention from earlier works. In SNNs, each Leaky-Integrate-and-Fire (LIF) neuron needs a specific memory called the membrane potential in order to store temporal information and generate output spikes. We observe that membrane potentials begin to take up a larger memory footprint with decreased weight precision. Furthermore, the size of membrane potential grows dramatically when using mini-batches of inputs. The proportion of the membrane potential (32-bit) in the total memory footprint increases from less than 20% to more than 60% on the VGG-9 SNN, as the weight precision drops from 32-bit to 4-bit, as shown on the left of Fig. 1. Furthermore, almost 98% of the overall memory footprint is taken up by membrane potentials at batch size 32.

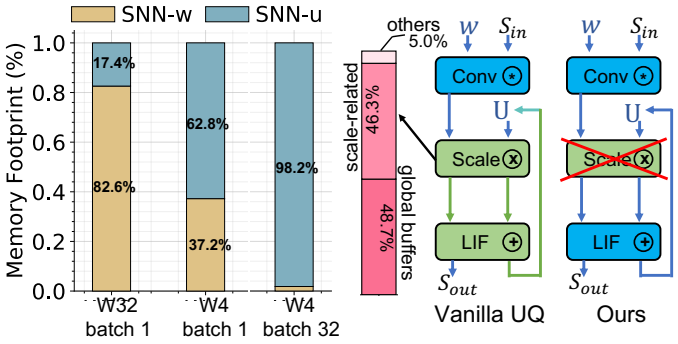

Fig. 1: Left: The proportion of membrane potential (u) in the total memory footprint of SNNs when reducing weight (w) precision and increasing mini-batches. Right: Comparison between our MINT quantization and the vanilla uniform quantization (UQ). We also show the breakup of the area cost for a 4-bit vanilla UQ SNN on SpinalFlow [9]. The color green denotes 32-bit operations while blue represents operations that scale with operand sizes.

Secondly, the scaling factor in uniform quantization (UQ) [10], [11] imposed by earlier SNN quantization works [6], [7] leads to substantial hardware overheads during deployment. In the vanilla UQ method, after the convolution between quantized weights and input spikes, the convolution result is multiplied by a full-precision (32-bit) scaling factor to ensure high inference accuracy as shown on the right of Fig. 1. Consequently, the systems that deploy SNNs physically require power and area-hungry multipliers, which leads to significant hardware overheads. For instance, SpinalFlow [9], an existing SNN accelerator, needs to allocate 46.3% of its system area to support the scaling factors for deploying a 4-bit UQ SNN model. In contrast, the convolution operations only account for 5% of the resources, as shown in Fig. 1. Unfortunately, ignoring the scaling factors in UQ will cause a significant accuracy degradation at inference time [11].

To address the above challenges, we propose Multiplier-less Integer-based (MINT) quantization. MINT is a uniform quantization scheme designed for both the weights and membrane potentials of SNNs. During training, we retain the scaling factors for MINT to ensure competitive accuracy. However, we show that by employing a shared scaling factor between weights and membrane potentials, we can eliminate the need for scaling factors during inference without sacrificing accu-

racy, and thus remove the 32-bit multipliers on the hardware. We have illustrated our approach on the right-hand side of Fig. 1. Our key contributions can be listed as follows:

1) We identify two crucial obstacles to SNN quantization: the memory-intensive nature of membrane potentials and the hardware-resource demands of scaling factors. To tackle these problems, we introduce the MINT quantization scheme, which quantizes both weights and membrane potentials to extremely low precisions (2-bit) and eliminates the need for scaling factors during inference without compromising accuracy.
2) Experimental results demonstrate that our approach outperforms other state-of-the-art methods and full-precision baselines in terms of total memory footprint at iso-accuracy. For example, compared to the full-precision model, our 2-bit quantized VGG-16 model on CIFAR-10 achieves a remarkable 93.8% reduction in memory footprint with only a minuscule accuracy degradation of 0.6%.
3) We designed an SNN accelerator in 32nm CMOS technology for comparing the hardware performances of MINT with the vanilla UQ method. Additionally, we evaluated our method on two existing specialized SNN accelerators, SpinalFlow [9] and PTB [12]. The results show that MINT significantly saves hardware resources, with an average of 85% reduction in PE-array area and 90% reduction in computation energy compared to earlier techniques, making it an edge-friendly approach.

II. Related Work

Quantization for SNNs has been extensively studied by prior works [6]–[8], [13]–[19]. Depending on the training scheme of the SNNs, those works can be categorized into three groups.

BPTT-based SNNs In [7], the ADMM method optimized a pre-trained full-precision network to quantize the weights into low precision. Meanwhile, [6] used K-means clustering quantization to achieve reasonable accuracy with 5-bit weight SNNs. A recent work [8] explored fully integer-based SNN training with weights, membrane potentials (ranging from 8-bit to 16-bit), and gradients quantized into the fixed-point format. Our MINT method also falls into this category, although we get rid of scaling factors while reducing both weights and membrane potentials to extremely low precision (up to 2-bit).

STDP-based SNNs Recent works on weight quantization of SNNs [13]–[17] have primarily focused on shallow networks with local spike timing dependent plasticity (STDP) learning, which does not scale for complex image classification tasks.

Conversion-based SNNs Several works investigate quantization in the ANN-to-SNN conversion technique. Those works put their efforts into compressing the ANN activations to get better energy and accuracy performance on the converted-SNNs [18], [19] while keeping the weights and membrane potentials at full or integer precision. [double]

Additionally, prior efforts have explored pruning methods to reduce the SNN sizes [6], [7]. Further, certain works have proposed reducing the number of timesteps required for processing an input which translates to energy and memory savings on hardware [4]. MINT is orthogonal and complementary to these SNN-optimization schemes. Finally, there have been separate hardware efforts to accelerate SNNs during inference with temporal/spatial reuse dataflows of weights and membrane potentials [9], [12], [20]. MINT can be deployed on these accelerators without additional hardware resources.

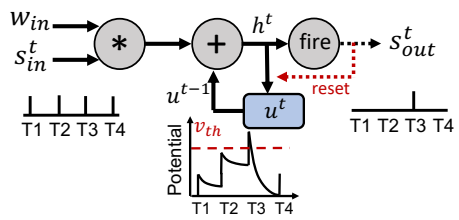

Fig. 2: Illustration of the behavior of the LIF neuron.

III. Preliminaries

Spiking LIF Neurons SNNs process unary spike trains over multiple discrete timesteps using Leaky-Integrate-and-Fire (LIF) neurons. These neurons introduce non-linearity and capture temporal information in the network. The behavior of LIF neurons during the inference time can be described in three stages. First, the 'update' stage updates the membrane potential matrix at layer l, denoted as $\boldsymbol{H}_l^{(t)}$, using the weight matrix $\boldsymbol{W}_l$ and the input spike matrix $\boldsymbol{S}_{l-1}^{(t)}$ from the previous layer at timestep t. Additionally, it aggregates the residual membrane potential matrix from the previous timestep, denoted as $\boldsymbol{U}_l^{(t-1)}$:

$$\boldsymbol{H}_l^{(t)} = \boldsymbol{W}_l \boldsymbol{S}_{l-1}^{(t)} + \tau \boldsymbol{U}_l^{(t-1)}, \tag{1}$$

where $\tau \in (0, 1]$ is the leakage factor that mimics the decay of potential. Next, in the 'firing' stage, the output spike matrix $\boldsymbol{S}_l^{(t)}$ is computed by comparing the membrane potential $\boldsymbol{H}_l^{(t)}$ with a predefined firing threshold v_{th}:

$$\boldsymbol{S}_l^{(t)} = \begin{cases} 1 & \boldsymbol{H}_l^{(t)} > v_{th} \\ 0 & \text{else.} \end{cases} \tag{2}$$

Finally, in the 'reset' stage, the residual membrane potential will be reset to 0 if the output spike is 1, *i.e.* $\boldsymbol{U}_l^{(t)} = \boldsymbol{H}_l^{(t)}(1 - \boldsymbol{S}_l^{(t)})$. Fig 2 shows the behavior of one LIF neuron. Throughout the paper, we will refer to the quantization of membrane potential as the residual membrane potential, denoted by $\boldsymbol{U}$.

During training, we leverage the recently proposed surrogate gradient method to approximate the gradients for the non-differentiable LIF neuron [3]. We use the cut-off approximation following the previous work [4] to perform BPTT training for SNNs.

Uniform Quantization Uniform integer quantization (UQ) methods have been extensively studied by prior works [10], [11]. UQ involves an affine mapping between the integer number vector $\mathbf{q}$ and the floating-point number vector $\mathbf{r}$, defined as follows:

$$\mathbf{r} = \alpha\,(\hat{\mathbf{q}} - Z)\,, \tag{3}$$

where α represents the scaling factor, and Z is the zero-point, both of which are the quantization hyperparameters. In n-bit quantization, the vector $\hat{\mathbf{q}}$ contains integers representing one of the 2^n quantized levels in the range of $[0, 2^n]$. The

scaling factor α is a 32-bit floating-point number used to scale the quantized levels to best match the distribution of the original values in $\mathbf{r}$. The zero-point Z is an integer that ensures $\mathbf{r} = 0$ can be exactly represented. Interestingly, we found that excluding the zero-point does not impact the final accuracy of the quantized SNN models. Hence, we do not consider the zero-point Z in the rest of this work.

IV. Multiplier-less Integer Quantization

A. Transform the LIF Equations

We start the transformation by naively applying UQ to Eq. 1 with distinct full-precision scaling factors α_1, α_2, and α_3 for each integer quantity. Assume there is no output spike fired at timestep t. Then we will have the following equation for the 'reset' stage:

$$\alpha_3 \hat{\boldsymbol{U}}_l^{(t)} = \alpha_1 \hat{\boldsymbol{W}}_l \boldsymbol{S}_{l-1}^{(t)} + \alpha_2 \tau \hat{\boldsymbol{U}}_l^{(t-1)}, \tag{4}$$

which can be rewritten as

$$\hat{\boldsymbol{U}}_l^{(t)} = \frac{\alpha_1}{\alpha_3} \hat{\boldsymbol{W}}_l \boldsymbol{S}_{l-1}^{(t)} + \frac{\alpha_2}{\alpha_3} \tau \hat{\boldsymbol{U}}_l^{(t-1)}. \tag{5}$$

In Eq. 5, the only two non-integers left are α_1/α_3 and α_2/α_3, assuming $\tau = 0.5$ (which can be computed by right shift). In order to remove those two full-precision multiplications, we assume $\alpha_1 = \alpha_2 = \alpha_3 = \alpha$. By having all the scaling factors equal to each other, we manage to transform Eq. 5 into:

$$\hat{\boldsymbol{U}}_l^{(t)} = \hat{\boldsymbol{W}}_l \boldsymbol{S}_{l-1}^{(t)} + \tau \hat{\boldsymbol{U}}_l^{(t-1)}. \tag{6}$$

Eq. 6 now only consists of integer values and integer operations. Moreover, there is no multiplication required for the scaling factor. We again naively apply UQ into Eq. 2 and find that the 'firing' stage will only generate an output spike 1 if the following inequality holds true:

$$\alpha_1 \hat{\boldsymbol{W}}_l \boldsymbol{S}_{l-1}^{(t)} + \alpha_2 \tau \hat{\boldsymbol{U}}_l^{(t-1)} > v_{th}. \tag{7}$$

Since $\alpha_1 = \alpha_2 = \alpha$, we can divide both sides of the inequality by α. The Eq. 7 is thus transformed to the form of:

$$\hat{\boldsymbol{W}}_l \boldsymbol{S}_{l-1}^{(t)} + \tau \hat{\boldsymbol{U}}_l^{(t-1)} > \frac{v_{th}}{\alpha}. \tag{8}$$

We have empirically observed that α is always greater than zero, which means that the direction of the inequality in Eq. 8 is always preserved. Additionally, since the left-hand side of Eq. 8 is always an integer, we can transform Eq. 8 into the following form and still generate the same output spikes:

$$\hat{\boldsymbol{W}}_l \boldsymbol{S}_{l-1}^{(t)} + \tau \hat{\boldsymbol{U}}_l^{(t-1)} \geq \left\lceil \frac{v_{th}}{\alpha} \right\rceil, \tag{9}$$

where $\lceil \cdot \rceil$ is the ceil operation. We define this new integer firing threshold $\lceil \frac{v_{th}}{\alpha} \rceil$ as θ, a constant that can be computed offline. Empirically we find that it is enough to use less than 6 bits to represent θ for each layer. We will use Eq. 6 and Eq. 9 as the new LIF equations for our MINT method during inference.

Algorithm 1 Inference path at l-th layer of the MINT-quantized SNN at timestep t. The leakage factor $\tau = 0.5$.

Input:
Input spikes $\boldsymbol{S}_{l-1}^{(t)}$ to the layer l at timestep t,
integer weights $\hat{\boldsymbol{W}}_l$ of layer l,
integer membrane potential $\hat{\boldsymbol{U}}_l^{(t-1)}$ of layer l at timestep $t-1$,
integer firing threshold θ of value $\lceil \frac{v_{th}}{\alpha} \rceil$.
Output:
Output spikes $\boldsymbol{S}_l^{(t)}$ to the layer $l+1$ at timestep t,
integer membrane potential $\hat{\boldsymbol{U}}_l^{(t)}$ of layer l at timestep t.

$\boldsymbol{X}_l^{(t)} \leftarrow \hat{\boldsymbol{W}}_l \boldsymbol{S}_{l-1}^{(t)}$
$\boldsymbol{H}_l^{(t)} \leftarrow \boldsymbol{X}_l^{(t)} + \boldsymbol{U}_l^{(t-1)} >> 1$
if $\boldsymbol{H}_l^{(t)} \geq \theta$ **then**
    $\boldsymbol{S}_l^{(t)} \leftarrow 1$
    $\hat{\boldsymbol{U}}_l^{(t)} \leftarrow 0$
else
    $\boldsymbol{S}_l^{(t)} \leftarrow 0$
    $\hat{\boldsymbol{U}}_l^{(t)} \leftarrow \boldsymbol{H}_l^{(t)}$
end if

B. Inference Datapath

As discussed in Sec. IV-A, by sharing the scaling factor between weights and membrane potentials across timesteps for each layer, we can implement our quantization scheme using integer-only arithmetic that requires no multiplications during inference.

The inference algorithm is described in Algorithm 1. Firstly, a convolution operation is performed between the input spikes and weights. Given the unary nature of the input spikes and the quantized integer weights, the convolution operation simplifies into an integer-based accumulation. Subsequently, the integer convolution result $\boldsymbol{X}_l^{(t)}$ is added to the quantized integer residual membrane potential $\boldsymbol{U}_l^{(t-1)}$ to compute the membrane potential $\boldsymbol{H}_l^{(t)}$. The $\boldsymbol{U}_l^{(t-1)}$ undergoes a right shift by 1, which is equivalent to multiplication by a leakage factor with a value of 0.5. Next, the integer membrane potential $\boldsymbol{H}_l^{(t)}$ is compared with the pre-defined integer firing threshold θ, as discussed in Sec. IV-A. Based on the comparison results, the unary output spikes are generated. If no output spike is generated, the integer $\boldsymbol{H}_l^{(t)}$ will be stored as the residual membrane potential $\boldsymbol{U}_l^{(t)}$. Conversely, if an output spike is generated, a zero is stored as the residual membrane potential.

This inference path is repeated for all other layers and timesteps in SNNs, utilizing integer arithmetic exclusively, and effectively eliminating the need for multiplications.

C. Training with Quantization

During the forward path of training, we apply the quantization function (see Eq. 10) to both weights and membrane potentials. As discussed in Sec. IV-A, our method relies on the assumption that $\alpha_1 = \alpha_2 = \alpha_3 = \alpha$. To achieve this, we introduce a dummy scaling factor α for each layer, which can be shared between weights and membrane potentials. Consequently, the following quantization function Q is applied during training:

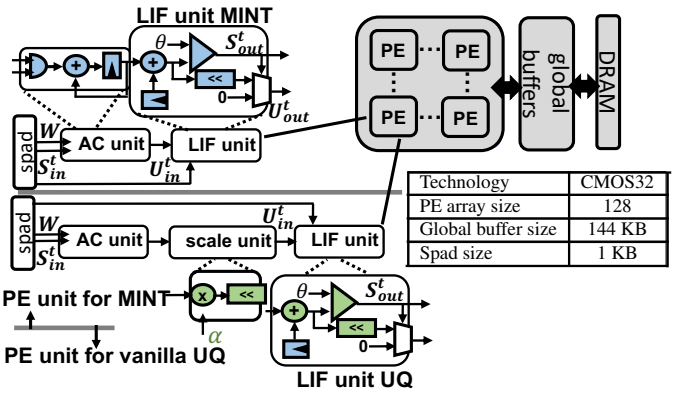

Fig. 3: Systolic-array-based architecture with PE that deploys vanilla UQ models that require scaling factors and MINT quantized models that do not require scaling factors. Blue units can be scaled with the operand sizes and green units are fixed to 32-bit.

$$Q(r,n) = \frac{\lfloor \text{clamp}\left(\frac{r}{\alpha}, -1, 1\right)\text{s}(n) \rceil\, \alpha}{\text{s}(n)}, \tag{10}$$

$$\text{s}(n) = 2^{n-1} - 1,$$
$$\text{clamp}(r, a, b) = \min(\max(r, a), b).$$

Here, r represents a full-precision floating-point number to be quantized, and n represents the number of bits assigned to the quantized integer. As we focus on uniform quantization in this work, we maintain the same n for both the weights and membrane potentials across all layers and timesteps.

During backward propagation, we use full-precision floating-point gradients for all quantities. To approximate the derivative for the non-differentiable rounding function $\lfloor \cdot \rceil$, we employ the straight-through estimator [10].

Furthermore, instead of assigning scaling factors for weights and membrane potentials as hyperparameters that do not change, we make α a learnable parameter. For each layer, a distinct learnable full-precision scaling factor α is introduced. The initialization of α is set to $\frac{2<|w|>}{\text{s}(n)}$, where w denotes the initial weight values. $< \cdot >$ and $|\cdot|$ represent the arithmetic-mean and absolute value respectively. During backward propagation, we update α with full-precision gradients. Empirically, we observe that scaling the loss gradient of α by $\frac{1}{\sqrt{N_w \text{s}(n)}}$ achieves the fastest convergence and best accuracy. Here N_w represents the total number of weights in each layer.

V. SNN Accelerator Design for Quantization

In recent systolic array-based SNN inference accelerators [9], [12], [20], low-precision of weights and membrane potentials have been assumed to save memory and computation unit costs. However, prior designs normally neglect to include the scaling units for supporting the vanilla UQ models. To better compare the hardware performance between MINT and vanilla UQ at inference time, we design and implement a systolic-array-based SNN accelerator in Fig. 3. The accelerator adopts the temporal output-stationary dataflow in [9], [20], where the accumulation (AC) and LIF operation for each neuron stays stationary in a processing element (PE) across all timesteps. We adopt three levels of the memory hierarchy: 1) an off-chip DRAM, 2) a global buffer, and 3) a scratch pad.

We further provide two PE designs for two methods. For running the MINT models, each PE consists of 1) an AC unit to perform the 'update' stage, and 2) an LIF unit to perform the 'firing' and 'reset' stage. All the arithmetic units in MINT PE are blue, denoting that their size can be scaled down with the precision of weights and membrane potentials. For running the UQ models, a scaling unit consists of a 32-bit multiplier, and a shifter is added between the AC unit and the LIF unit. Moreover, the LIF unit in UQ PE requires a 32-bit operand size for adapting the full-precision input from the scaling unit. All the units denoted as green have 32-bit precision. Both PE design has two scratch pad memory to hold the weights and input spikes. A small register is equipped inside the LIF unit to store the residual membrane potential. All the arithmetic units in both PE designs are integer-based. We allocate 144 KB for the global buffer and 1 KB for the scratch pad memory inside each PE. And we set the PE array size to 128.

TABLE I: Accuracy results of different precision of MINT in comparison to the full-precision baselines.

Method	Precision (W - U) 8 - 8	4 - 4	2 - 2
VGG-9 (CIFAR-10)	Top1-Accuracy (%)		
`fp32`: 88.03			
MINT (Ours)	87.48	87.37	87.47
VGG-16 (CIFAR-10)	Top1-Accuracy (%)		
`fp32`: 91.15			
MINT (Ours)	90.72	90.65	90.56
ResNet-19 (CIFAR-10)	Top1-Accuracy (%)		
`fp32`: 91.29			
MINT (Ours)	91.36	91.45	90.79
VGG-16 (TinyImageNet)	Top5-Accuracy (%)		
`fp32`: 73.71			
MINT (Ours)	73.92	73.33	73.18
ResNet-19 (CIFAR10-DVS)	Top5-Accuracy (%)		
`fp32`: 94.2			
MINT (Ours)	94.4	94.5	93.7

VI. Experiments

A. Experimental Settings

Algorithm Setups. We choose three representative deep network architectures: VGG-9 [21], VGG-16 [21], and ResNet-19 [22]. We evaluate our quantization schemes on two static visual datasets: CIFAR-10 [23] and TinyImageNet [5] and one event-based dataset CIFAR-10 DVS [24]. We compare our MINT method with the full-precision baseline (`fp32`), *i.e.*, the model trained with 32-bit floating-point weights and membrane potentials. The training method for the MINT and the full-precision baseline is kept the same, except that MINT applies the quantization function in Eq. 10 during the forward propagation. We train our SNNs using the direct encoding technique [2], which has demonstrated remarkable effectiveness in training SNNs within a few timesteps. Unless otherwise specified, all experiments in the following sections use timesteps T = 4 ($T = 8$ for CIFAR10-DVS) and a batch size of 128. Adam optimizer with a learning rate of 0.001 is used. All models and training codes are implemented in PyTorch, as done in prior work [4].

Hardware Setups. We synthesize our accelerator in Sec. V using Synopsys Design Compiler at 400MHz using 32nm CMOS technology. We use CACTI [25] to simulate on-chip SRAM and off-chip DRAM to obtain memory statistics. The energy results are generated from SATASim, a cycle-accurate SNN energy simulator [20].

B. Experimental Results

Accuracy. Table I summarizes the accuracy results. We test MINT with 3 groups of bit-width, *i.e.*, 2,4,8. For instance, `W8U8` represents an SNN with weights and membrane potentials quantized to 8-bit integers. Although the full-precision baseline achieves higher accuracy for most of the datasets and networks, MINT achieves very close accuracy for all cases with less than a 1% drop. Some of our `W8U8` and `W4U4` models even have higher accuracy than the full-precision baseline (*e.g.,* 0.2% on VGG-16 TinyImageNet), which suggests the MINT may serve the purpose of regularization.

Memory Saving. In this section, we first present the memory footprint reduction with MINT with a batch size of 1. As shown in Fig. 4 (Left), our `W2U2` models on average give us more than 93% reduction in the total memory footprint. However, to improve inference speed, prior SNN works [2], [3], [26] commonly use mini-batch sizes larger than 1. In such cases, membrane potential quantization is critical. As illustrated in Fig. 4 (Right), the membrane potential overhead becomes sizeable with increasing batch size. We find that for a MINT quantized VGG-16 model on TinyImageNet with a batch size of 16, the compression of weight from 32-bit to 4-bit alone reduces the total memory footprint by 15%. Whereas, quantizing the membrane potential to 4 bits can further reduce 72.4% of the total memory footprint. We can expect that a larger batch size (> 64) will amplify the benefits from membrane potential quantization.

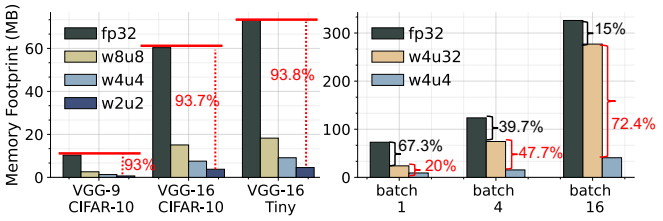

Fig. 4: Comparison of total memory footprint between full-precision and MINT-quantized models. For the left figure, we show the overall memory reduction with a batch size of 1. For the right figure, we further show the portion of memory reduction on different batch sizes.

Energy Saving. We study the inference energy difference between MINT and the vanilla UQ method on our proposed accelerator design in Sec. V. We normalize the results with the energy cost of a 16-bit integer multiply-accumulate operation. As illustrated in Fig. 5, the computation energy of the UQ model hardly reduces with decreasing operand size, due to the power-hungry full-precision multipliers for scaling factors. On the other hand, as our MINT quantized model does not require multipliers inside PEs, the computation energy scales down significantly with the operands' precision. Compared to the vanilla UQ method, MINT spends on average 90% less computation energy. We also show the memory energy

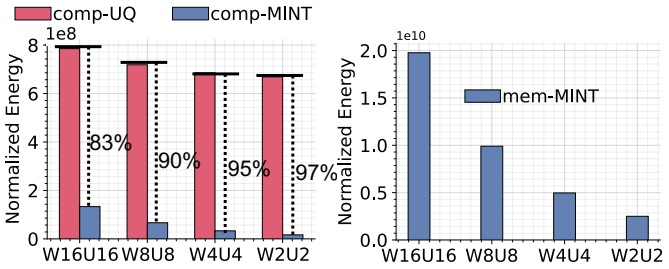

Fig. 5: Left: Normalized computation energy cost between MINT and vanilla UQ on VGG-9 CIFAR-10. Right: Normalized memory energy cost of MINT across different operand precisions.

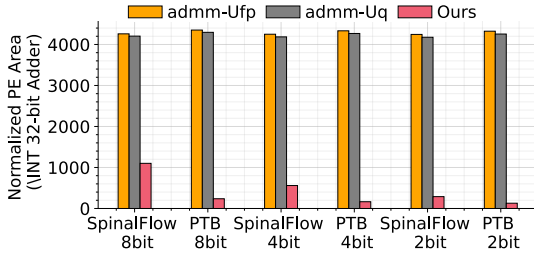

Fig. 6: Comparisons of the area between MINT and prior SNN quantization methods that use scaling factors. The weight precisions are indicated in the figure.

reduction brought by the quantization in Fig. 5. Our MINT-quantized `W2U2` VGG-9 network achieves an ∼87.3% memory energy reduction on CIFAR-10 compared to the `W16U16` model.

Comparison to Prior Works. We compare our quantization scheme with several state-of-the-art BPTT-based SNN quantization works on the CIFAR-10 dataset, including STBP-Quant [8], ST-Quant [6], and ADMM-Quant [7]. We aligned the weight precision, number of mini-batches, and number of timesteps ($T = 8$) for all methods. As shown in Table II, our method is the first to consider compressing both the weights and membrane potentials to a very low precision (< 8 bits) while yielding iso-accuracy with prior works.

TABLE II: Accuracy and total memory footprint comparison to prior state-of-the-art SNN quantization work on CIFAR-10.

Method (CIFAR-10)	Precision (W / U)	Accuracy (%) Top-1	Mini Batches	Memory Footprint (MB)
STBP-Quant	8 / 14	86.65	50	353.79
MINT (Ours)	8 / 8	**88.25**	50	**95.41**
ST-Quant	5 / 32	**88.6**	32	751.04
MINT (Ours)	5 / 5	88.04	32	**59.62**
ADMM-Quant	4 / 32	**89.4**	50	1279.66
STBP-Quant	4 / 10	84.99	50	248.39
MINT (Ours)	4 / 4	88.12	50	**47.71**
ADMM-Quant	2 / 32	**89.23**	50	1264.85
STBP-Quant	2 / 8	33.53	50	195.68
MINT (Ours)	2 / 2	88.39	50	**23.85**

We further demonstrate that compared to prior quantization works that rely on scaling factors, our MINT method is agnostic to the hardware design at deployment time and thus brings no hardware overheads. We synthesize two existing

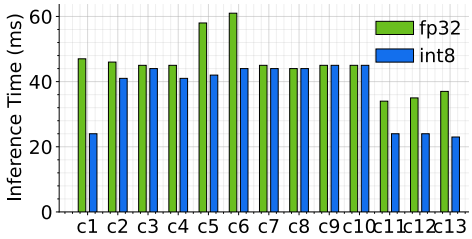

Fig. 7: Layer-wise speedup results for VGG-16 on TinyImageNet. Results are shown for A100 GPUs.

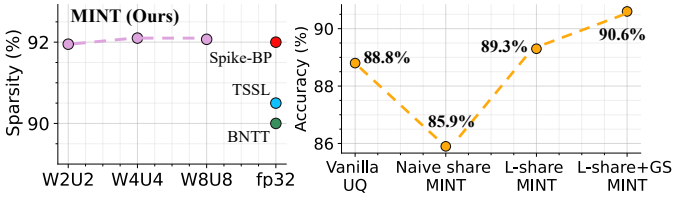

Fig. 8: Left: Average spike sparsity for MINT compared with other full precision SNN works, including BNTT [26], TSSL [3], and Spike-BP [27]. Right: Final test accuracy across different methods.

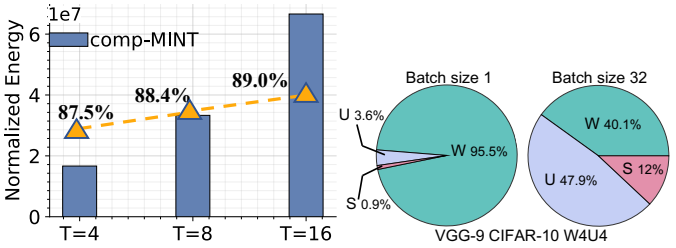

Fig. 9: Left: Tradeoffs between accuracy and normalized computation energy cost across timesteps. Accuracy is denoted by the yellow line plot. Right: Breakup of the memory energy. U denotes the membrane potentials, W denotes the weights, and S denotes the output spikes.

SNN inference accelerators SpinalFlow [9] and PTB [12] with two PE designs that support MINT and ADMM-Quant as in Sec. V. Fig. 6 shows the PE-array level area differences. Compared to the original ADMM-Quant that has `fp32` membrane potentials (`admm-Ufp`), MINT saves 76% (93%), 86% (95%), and 93% (96%) PE-array area with weights of 8 bits, 4 bits, and 2 bits on SpinalFlow (PTB), respectively. We further compare the ADMM-Quant method with membrane potentials at the same precision as the weights (`admm-Uq`). It turns out that the trend of area reduction still holds since ADMM-Quant relies on scaling factors.

C. Ablation Studies

In Fig. 7, we present the layerwise inference latency of our `W8U8` MINT model compared to the full-precision baseline for VGG-16 on TinyImageNet, implemented by CuDNN and tested on NVIDIA A100. Specifically, we find that the `W8U8` model consistently runs faster than the full-precision baseline on the first two layers, while also demonstrating a significant acceleration on the last three layers. MINT achieves an overall acceleration of 17.4% for TinyImageNet on A100.

On the left side of Fig. 8, we show the average spike sparsity of our MINT-quantized VGG-9 models on CIFAR-10 and other state-of-the-art full-precision SNN works. The results show that our method does not incur any sparsity reduction across different model precisions. By maintaining the same level of high spike sparsity as other SNN works, our quantized models can enjoy the same amount of computation energy reduction [20], [26]. On the right side of Fig. 8, we show the final test accuracy of VGG-16 on CIFAR-10 between the vanilla UQ and the MINT with different optimization techniques applied. In the Naive share MINT, we use a pre-determined scaling factor for weights and membrane potentials, which degrades the accuracy from the vanilla UQ by nearly 3%. We then make the shared scaling factors learnable (L-share MINT) which increases the accuracy by 3.4%. Finally, scaling the gradient of the learnable shared scaling factors (L-share+GS) as discussed in Sec. IV-C contributes to another 1.3% accuracy increase.

On the left side of Fig. 9, we further use the `W2U2` VGG-9 model on CIFAR-10 to show that our method works with different timesteps. While increasing timesteps will bring slightly higher accuracy, the computation energy will scale up linearly with the timesteps. Finally, we showcase the breakup of the memory energy in Fig. 5 in Fig. 9 (Right). The result matches the prior observation on membrane potential quantization. While the data movement energy of weights dominates at a batch size of 1, the memory energy cost of membrane potential increases to 47.9% of the total memory energy at a batch size of 32, justifying membrane potential quantization.

VII. Conclusion

In this paper, we have introduced multiplier-less integer-based (MINT) quantization for both the weights and the membrane potentials of SNNs. By sharing the quantization scale between the weights and membrane potentials and transforming the LIF update equations, we are able to significantly compress the memory footprint of SNNs while dodging all the hardware overheads brought by scaling factors in vanilla quantization. Our method achieves comparable accuracy to full-precision baselines and other state-of-the-art SNN quantization methods. In conclusion, our proposed MINT quantization method offers a practical and effective solution for reducing the memory footprint and hardware resource requirements of SNNs without sacrificing accuracy, opening up new possibilities for SNN-based edge computing.

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, Google Research Scholar Award, the National Science Foundation CAREER Award, TII (Abu Dhabi), the DARPA AI Exploration (AIE) program, 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*, 2019.
[2] Y. Wu *et al.*, "Direct training for spiking neural networks: Faster, larger, better," in *AAAI*, 2019.
[3] W. Zhang *et al.*, "Temporal spike sequence learning via backpropagation for deep spiking neural networks," *NeurIPS*, 2020.
[4] Y. Li *et al.*, "Input-aware dynamic timestep spiking neural networks for efficient in-memory computing," *DAC*, 2023.

[5] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," *CVPR*, 2009.

[6] S. S. Chowdhury *et al.*, "Spatio-temporal pruning and quantization for low-latency spiking neural networks," in *IJCNN*, 2021.

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

[8] P.-Y. Tan *et al.*, "A low-bitwidth integer-stbp algorithm for efficient training and inference of spiking neural networks," in *ASPDAC*, 2023.

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

[10] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018.

[11] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv:1806.08342*, 2018.

[12] J.-J. Lee *et al.*, "Parallel time batching: Systolic-array acceleration of sparse spiking neural computation," in *HPCA*, 2022.

[13] H. W. Lui *et al.*, "Hessian aware quantization of spiking neural networks," in *ICONS*, 2021.

[14] C. J. Schaefer *et al.*, "Quantizing spiking neural networks with integers," *ICONS*, 2020.

[15] N. Rathi *et al.*, "Stdp-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition," *TCAD*, 2018.

[16] S. Hu *et al.*, "Quantized stdp-based online-learning spiking neural network," *Neural Computing and Applications*, 2021.

[17] R. V. W. Putra *et al.*, "Q-spinn: A framework for quantizing spiking neural networks," in *IJCNN*, 2021.

[18] A. R. Voelker *et al.*, "A spike in performance: Training hybrid-spiking neural networks with quantized activation functions," *arXiv:2002.03553*, 2020.

[19] C. Li *et al.*, "Quantization framework for fast spiking neural networks," *Frontiers in Neuroscience*, 2022.

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

[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 tinyimages," 2009.

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

[25] N. Muralimanohar *et al.*, "Cacti 6.0: A tool to model large caches," *HP laboratories*, 2009.

[26] Y. Kim *et al.*, "Revisiting batch normalization for training low-latency deep spiking neural networks from scratch," *Frontiers in neuroscience*, 2021.

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