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, a uniform quantization scheme that efficiently compresses weights and membrane potentials in spiking neural networks (SNNs). Unlike previous SNN quantization methods, MINT quantizes memory-intensive membrane potentials to an extremely low precision (2-bit), significantly reducing the memory footprint. MINT also shares the quantization scaling factor between weights and membrane potentials, eliminating the need for multipliers required in conventional uniform quantization. Experimental results show that our method matches the accuracy of full-precision models and other state-of-the-art SNN quantization techniques while surpassing them in memory footprint reduction and hardware cost efficiency at deployment. For example, 2-bit MINT VGG-16 achieves 90.6% accuracy on CIFAR-10, with roughly 93.8% reduction in memory footprint from the full-precision model and 90% reduction in computation energy compared to vanilla uniform quantization at deployment.[1]

I. Introduction

Spiking Neural Networks (SNNs) [1] present a promising alternative to Artificial Neural Networks (ANNs), excelling in energy efficiency by processing sparse unary spike trains (0,1) across discrete timesteps. This efficiency makes SNNs ideal for low-power edge devices, as they operate with 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 led to inflated model sizes for SNNs to maintain competitive accuracy, making them unsuitable for memory-constrained edge devices. While recent efforts have explored compressing BPTT-based SNNs through quantization [6]–[8], specifically reducing weight memory size, two critical issues remain overlooked.

Firstly, prior works have not adequately addressed the size of the membrane potential in SNNs. Each Leaky-Integrate-and-Fire (LIF) neuron in an SNN requires a specific memory, known as the membrane potential, to store temporal information and generate output spikes. We observe that as weight precision decreases, membrane potentials begin to occupy a larger portion of the memory footprint, especially when using mini-batches of inputs. For instance, the proportion of the 32-bit membrane potential in the total memory footprint of the VGG-9 SNN increases from under 20% to over 60% as weight precision drops from 32-bit to 4-bit, as depicted on the left of Fig. 1. Further, membrane potentials account for nearly 98% of the overall memory footprint at a batch size of 32.

[1]Code is available at https://github.com/Intelligent-Computing-Lab-Yale/MINT-Quantization

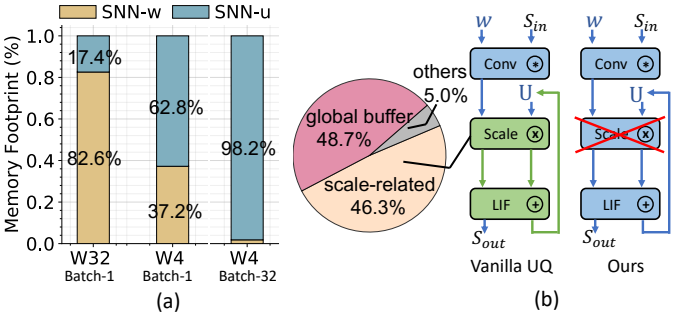

Fig. 1: (a): Proportion of membrane potential (u) in the total memory footprint of SNNs, varying with weight (w) precision and mini-batch size. (b): Comparison of our MINT quantization with vanilla Uniform Quantization (UQ), including a breakdown of area cost for a 4-bit vanilla UQ SNN on SpinalFlow [9]. Green indicates fixed 32-bit operations, while blue denotes quantized operations scalable with operand sizes.

Secondly, the use of a scaling factor in Uniform Quantization (UQ) [10], [11] by previous SNN quantization works [6], [7] results in substantial hardware overheads during deployment. In the vanilla UQ method, the convolution result obtained by convolving quantized weights with input spikes, is multiplied by a full-precision (32-bit) scaling factor to maintain high inference accuracy, as depicted in Fig. 1(b). This necessitates power and area-intensive multipliers in systems deploying SNNs, leading to significant hardware overheads. For example, SpinalFlow [9], an SNN accelerator, allocates 46.3% of its system area to support scaling factors for a 4-bit UQ SNN model, while convolution operations use only 5% of resources, as shown in Fig. 1. Omitting the scaling factors in UQ leads to substantial accuracy degradation at inference [11].

To address the aforementioned challenges, we introduce Multiplier-less INTeger-based (MINT) quantization, a uniform scheme for quantizing both weights and membrane potentials in SNNs. During training, MINT retains scaling factors to ensure competitive accuracy. However, during inference, we demonstrate that using a shared scaling factor for weights and membrane potentials eliminates the need for scaling factors, thereby removing 32-bit multipliers on the hardware, as illustrated in Fig. 1(b). Our key contributions are:

(1) We pinpoint two major hurdles in SNN quantization: the memory-intensive nature of membrane potentials and the hardware-resource demands of scaling factors. To address these, we propose MINT, which quantizes both weights and membrane potentials to extremely low precisions (2-bit) and removes the need for scaling factors during inference without sacrificing accuracy.

(2) Our experimental results show that MINT outperforms

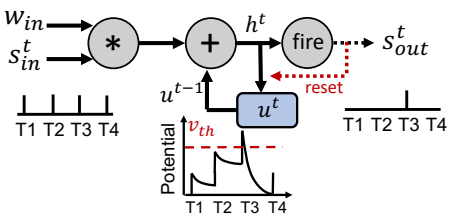

Fig. 2: Illustration of the behavior of the LIF neuron.

state-of-the-art methods and full-precision baselines in total memory footprint at iso-accuracy. For instance, our 2-bit quantized VGG-16 model on CIFAR-10 achieves a remarkable 93.8% reduction in memory footprint with a negligible accuracy degradation of 0.6%.

(3) We designed an SNN accelerator in 32nm CMOS technology to compare MINT's hardware performance with the vanilla UQ method. We also evaluated MINT on two existing SNN accelerators, SpinalFlow [9] and PTB [12]. Results indicate that MINT significantly conserves hardware resources, averaging an 85% reduction in PE-array area and a 90% reduction in computation energy compared to previous techniques, making it suitable for edge deployment.

II. Related Work

Quantization in SNNs has been thoroughly explored in previous studies [6]–[8], [13]–[19], which can be classified into three groups based on the SNNs' training scheme.

BPTT-based SNNs The ADMM method in [7] optimizes a pre-trained full-precision network to quantize weights to low precision. Meanwhile, [6] employs K-means clustering quantization to achieve reasonable accuracy with 5-bit weight SNNs. A recent study [8] investigates fully integer-based SNN training, with fixed-point quantized weights, membrane potentials (in 8-bit to 16-bit range), and gradients. Our MINT method also belongs to this category but eliminates scaling factors and reduces both weights and membrane potentials to extremely low precision (*i.e.* 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.

Additionally, prior efforts have explored pruning methods to reduce the SNN size [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.

III. Preliminaries

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

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

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

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

Lastly, in the '**reset**' stage, the residual membrane potential is 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 illustrates the behavior of an LIF neuron. Throughout the paper, we refer to the quantization of the membrane potential as the residual membrane potential, denoted by $\boldsymbol{U}$. For training, we use the surrogate gradient method to approximate gradients for the non-differentiable LIF neuron [3] and employ the cut-off approximation for BPTT training of SNNs [4].

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

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

where α is the scaling factor, and Z is the zero-point, both of which are quantization hyperparameters. In n-bit quantization, $\hat{\mathbf{q}}$ contains integers representing one of the 2^n quantized levels in the range $[0, 2^n]$. The scaling factor α, a 32-bit floating-point number, scales the quantized levels to closely match the original distribution in $\mathbf{r}$. The zero-point Z is an integer that ensures $\mathbf{r} = 0$ is precisely represented. Interestingly, we found that omitting the zero-point does not affect the accuracy of quantized SNN models. Hence, we exclude the zero-point Z in subsequent discussions.

IV. Multiplier-less Integer Quantization

A. Transform the LIF Equations

We begin by naively applying vanilla UQ to Eq. 1, using distinct full-precision scaling factors α_1, α_2, and α_3 for each integer quantity. Assuming no output spike is fired at timestep t, the '**update**' stage equation is:

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

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, assuming $\tau = 0.5$ (computable by right shift), the only non-integer multiplicands are α_1/α_3 and α_2/α_3. In order to remove these 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 consists only of integer values and operations, without requiring multiplication for the scaling factor. We again naively apply vanilla UQ to Eq. 2. We find that the '**firing**' stage generates an output spike of 1 only 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. Eq. 7 is thus transformed into 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, sharing the scaling factor between weights and membrane potentials across timesteps for each layer allows us to implement our quantization scheme using integer-only arithmetic, eliminating the need for multiplications during inference.

The inference algorithm is detailed in Algorithm 1. First, a convolution operation is performed between the input spikes and weights. Given the unary nature of the input spikes and the quantized integer weights, this convolution operation simplifies to an integer-based accumulation. Then, 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, equivalent to being multiplied by a leakage factor of 0.5. Subsequently, the integer membrane potential $\boldsymbol{H}_l^{(t)}$ is compared with the pre-defined integer firing threshold θ, as discussed in Sec. IV-A. The comparison determines the generation of unary output spikes. If no output spike is generated, the integer $\boldsymbol{H}_l^{(t)}$ is 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 MINT Quantization

In the training's forward path, we apply the quantization function (refer to Eq. 10) to both weights and membrane potentials. As mentioned in Sec. IV-A, our approach assumes $\alpha_1 = \alpha_2 = \alpha_3 = \alpha$. To realize 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:

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

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

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

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

Moreover, instead of assigning a static scaling factor for weights and membrane potentials as one of the hyperparameters, 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 $\frac{1}{\sqrt{N_w \text{s}(n)}}$ ensures rapid convergence and optimal accuracy. Here N_w denotes 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 weights and membrane potentials have been used to reduce memory and computation costs. However, these designs normally neglect to include the scaling units required for supporting vanilla UQ models. To facilitate a comprehensive comparison of hardware performance between MINT and vanilla UQ during inference, we design and implement a systolic-array-based SNN accelerator,

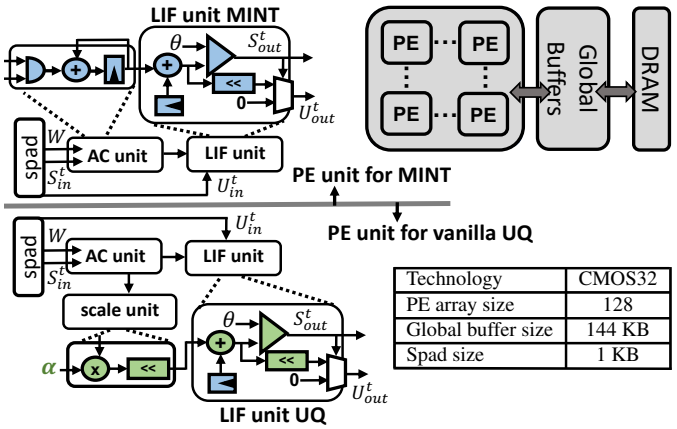

Fig. 3: Systolic-array-based architecture with PE for deploying vanilla UQ models, which require scaling factors, and MINT quantized models, which do not. Blue units are scalable with operand sizes, while green units are fixed at 32-bit.

as shown in Fig. 3. The accelerator adopts the temporal output-stationary dataflow from [9], [20], where the accumulation (AC) and LIF operations for each neuron remain stationary in a processing element (PE) across all timesteps. We adopt three levels of memory hierarchy: 1) an off-chip DRAM, 2) an SRAM-based global buffer, and 3) a register-file-based scratch pad.

We further provide two PE designs for the two methods. For running the MINT models, each PE includes 1) an AC unit to perform the '**update**' stage, and 2) an LIF unit for the '**firing**' and '**reset**' stage. All arithmetic units in the MINT PE are colored blue, indicating that their size can be scaled down with the precision of weights and membrane potentials. For running the vanilla UQ models, a scaling unit, comprising a 32-bit multiplier and a shifter (for approximating the 32-bit float multiplication using integer operation [10]), is added between the AC and LIF units. Additionally, the LIF unit in the UQ PE requires a 32-bit operand size to accomodate the full-precision input from the scaling unit. All units denoted in green have 32-bit precision. Both PE designs have two scratch pad memories for holding the weights and input spikes. A small register within the LIF unit stores the residual membrane potential. All 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 in each PE. And we set the PE array size to 128.

VI. Experiments

A. Experimental Settings

Algorithm Setups. We evaluate our quantization scheme using three representative deep network architectures: VGG-9 [21], VGG-16 [21], and ResNet-19 [22]. Our experiments involve 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 MINT and the full-precision baseline is identical, except that MINT applies the quantization function from Eq. 10 during forward propagation. We employ the direct encoding technique [2] for training SNNs, which has proven effective in training SNNs within a few timesteps. Unless stated otherwise, all experiments use timesteps $T = 4$ ($T = 8$ for CIFAR10-DVS) and a batch size of 128. We use the Adam optimizer with a learning rate of 0.001. All models and training codes are implemented in PyTorch, following previous work [4].

Hardware Setups. We synthesize the accelerator described in Sec. V using Synopsys Design Compiler at 400MHz using 32nm CMOS technology. We use CACTI7.0 [25] to model the on-chip SRAM and off-chip DRAM to obtain memory statistics. Energy results are generated using SATASim, a cycle-accurate SNN energy simulator [20].

B. Experimental Results

Accuracy. Table I summarizes the accuracy results. We evaluate MINT across three bit-width groups: 2, 4, and 8. For example, `W8U8` denotes an SNN with both weights and membrane potentials quantized to 8-bit integers. While the full-precision baseline generally outperforms in terms of accuracy across most datasets and networks, MINT remains competitive, showing less than 1% accuracy drop in all tested scenarios. Notably, some of our `W8U8` and `W4U4` models even surpass the full-precision baseline in accuracy (e.g., a 0.2% increase on VGG-16 TinyImageNet), which suggests MINT may serve the purpose of regularization.

TABLE I: Accuracy comparison between MINT and full-precision baselines across different precisions.

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

Memory Saving. In this section, we first present the memory footprint reduction achieved by MINT for a batch size of 1. As depicted in Fig. 4(a), our `W2U2` models, on average, yield over 93% reduction in total memory footprint. However, to enhance inference speed, previous SNN works [2], [3], [26] often employ mini-batch sizes greater than 1. In such scenarios, the quantization of membrane potential becomes critical. As shown in Fig. 4(b), the memory overhead of the membrane potential grows significantly with larger batch sizes. For instance, for a MINT-quantized VGG-16 model on TinyImageNet with a batch size of 16, merely compressing the weight from 32-bit to 4-bit results in a 15% reduction in total memory footprint. In contrast, further quantizing the membrane potential to 4 bits leads to an additional 72.4% memory footprint reduction. We anticipate even more pronounced benefits from membrane potential quantization with larger batch sizes (greater than 64).

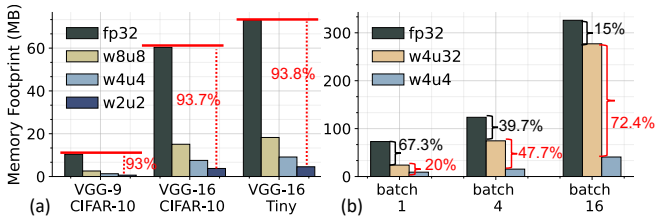

Fig. 4: Comparison of total memory footprint between full-precision and MINT-quantized models. (a) Overall memory reduction with a batch size of 1. (b) Memory reduction portion for different batch sizes.

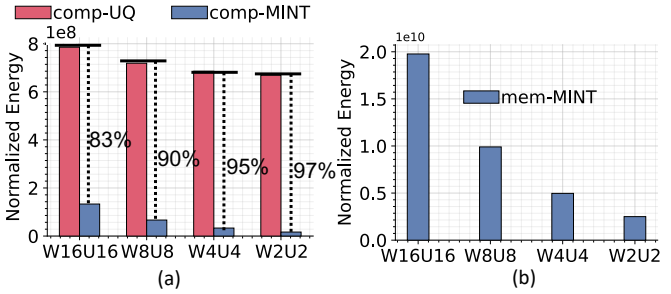

Fig. 5: (a) Normalized computation energy comparison between MINT and vanilla UQ on VGG-9 CIFAR-10. (b) Normalized memory energy cost of MINT for different operand precisions.

Energy Saving. We examine the difference in inference energy consumption between MINT and the vanilla UQ method using the accelerator design proposed in Sec.V. We normalize the results with the energy cost of a 16-bit integer multiply-accumulate operation. Fig. 5(a) demonstrates that the computation energy of the UQ model remains relatively constant despite reductions in operand size, due to the energy-intensive full-precision multipliers needed for scaling factors. Conversely, MINT-quantized models, which do not require multipliers within PEs, exhibit a substantial decrease in computation energy as operand precision decreases. On average, MINT consumes 90% less computation energy compared to the vanilla UQ method. Fig. 5(b) highlights the memory energy savings achieved through quantization. For instance, our MINT-quantized `W2U2` VGG-9 network attains an approximately 87.3% reduction in memory energy on CIFAR-10 compared to the `W16U16` model.

Comparison to Prior Works. We benchmark MINT against several state-of-the-art BPTT-based SNN quantization methods on the CIFAR-10 dataset, including STBP-Quant [8], ST-Quant [6], and ADMM-Quant [7]. We ensure consistency in weight precision, number of mini-batches, and number of timesteps ($T = 8$) across all methods. Table II reveals that our method is the first to explore compressing both weights and membrane potentials to extremely low precision (< 8 bits) while maintaining accuracy comparable to previous works.

We also showcase that, unlike previous quantization methods that depend on scaling factors, our MINT approach is agnostic to the hardware design at deployment time and incurs no additional hardware overheads. We synthesize two existing SNN inference accelerators SpinalFlow [9] and PTB [12], with two PE designs supporting both MINT and ADMM-Quant

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**

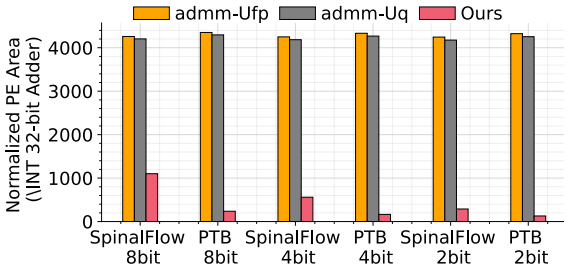

Fig. 6: Area comparison between MINT and prior SNN quantization methods using scaling factors.

as described in Sec. V. Fig. 6 illustrates the differences in PE-array level area. When compared to the original ADMM-Quant, which uses `fp32` membrane potentials (`admm-Ufp`), MINT achieves a reduction of 76% (93%), 86% (95%), and 93% (96%) in PE-array area with 8-bit, 4-bit, and 2-bit weights on SpinalFlow (PTB), respectively. We also compare ADMM-Quant with membrane potentials having the same precision as the weights (`admm-Uq`). The trend of area reduction persists, as ADMM-Quant still depends on scaling factors.

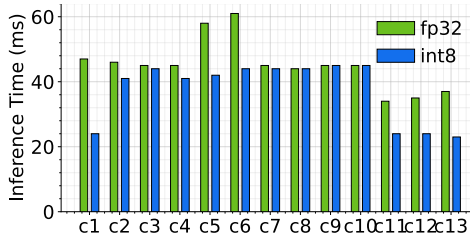

Fig. 7: Layer-wise speedup of VGG-16 on TinyImageNet on NVIDIA A100 GPU.

C. Ablation Studies

In Fig. 7, we compare the layerwise inference latency of our `W8U8` MINT model with the full-precision baseline for VGG-16 on TinyImageNet, implemented using CuDNN and tested on an NVIDIA A100. Notably, the `W8U8` model consistently outperforms the full-precision baseline in terms of speed on the first two layers and shows substantial acceleration on the last three layers. Overall, MINT achieves a 17.4% acceleration for TinyImageNet on the A100.

In Fig. 8(a), we display the average spike sparsity of our MINT-quantized VGG-9 models on CIFAR-10 alongside other state-of-the-art full-precision SNN works. The results

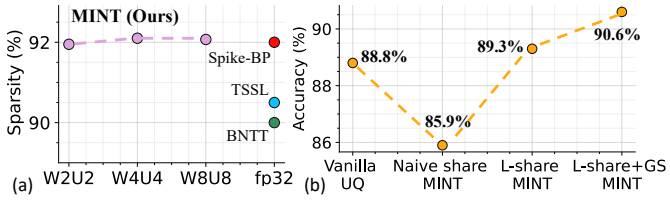

Fig. 8: (a) Average spike sparsity of MINT versus other full-precision SNN works: BNTT [26], TSSL [3], and Spike-BP [27]. (b) Final test accuracy comparison.

indicate that our method does not lead to any reduction in sparsity across different model precisions. By maintaining high spike sparsity comparable to other SNN works, our quantized models can benefit from similar computation energy reductions [20], [26]. In Fig. 8(b), we compare the final test accuracy of VGG-16 on CIFAR-10 between vanilla UQ and MINT with various optimization techniques applied. In the Naive share MINT, we use a predetermined scaling factor for weights and membrane potentials, resulting in a nearly 3% accuracy drop from the vanilla UQ. Making the shared scaling factors learnable (L-share MINT) recovers the accuracy by 3.4%. Additionally, scaling the gradient of the learnable shared scaling factors (L-share+GS), as discussed in Sec. IV-C, leads to a further 1.3% increase in accuracy.

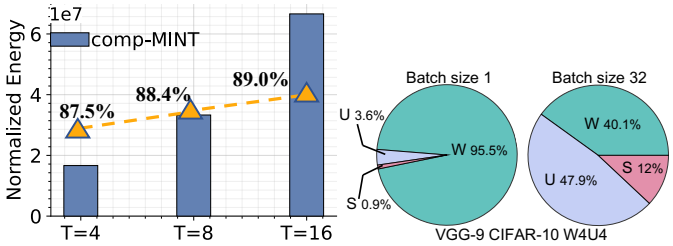

Fig. 9: (a) Tradeoffs between accuracy (yellow line) and normalized computation energy across timesteps. (b) Memory energy breakdown: U for membrane potentials, W for weights, and S for output spikes.

In Fig. 9(a), we demonstrate that our method is effective with different timesteps using the `W2U2` VGG-9 model on CIFAR-10. While increasing timesteps slightly improves accuracy, the computation energy linearly increases with the timesteps. Lastly, we present a breakdown of memory energy in Fig.9(b). The results align with previous observations on membrane potential quantization. While the data movement energy of weights dominates at a batch size of 1, the memory energy cost of the membrane potential rises to 47.9% of the total memory energy at a batch size of 32, underscoring the importance of membrane potential quantization.

VII. Conclusion

In this paper, we presented the multiplier-less integer-based (MINT) quantization method for SNNs, which compresses both weights and membrane potentials. By sharing the quantization scale and transforming the LIF update equations, MINT significantly reduces the memory footprint of SNNs without incurring hardware overheads associated with scaling factors in conventional quantization. Our approach achieves accuracy comparable to full-precision baselines and other state-of-the-art SNN quantization methods. In conclusion, MINT provides a practical solution for minimizing memory footprint and hardware requirements of SNNs without compromising accuracy, opening up new possibilities for efficient SNN-based edge computing.

Acknowledgments

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, A. Jaiswal, and P. Panda, "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 and P. Li, "Temporal spike sequence learning via backpropagation for deep spiking neural networks," *NeurIPS*, 2020.
[4] Y. Li, A. Moitra, T. Geller, and P. Panda, "Input-aware dynamic timestep spiking neural networks for efficient in-memory computing," *DAC*, 2023.
[5] J. Deng, "Imagenet: A large-scale hierarchical image database," *CVPR*, 2009.
[6] S. S. Chowdhury, I. Garg, and K. Roy, "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 and C.-W. Wu, "A low-bitwidth integer-stbp algorithm for efficient training and inference of spiking neural networks," in *ASPDAC*, 2023.
[9] S. Narayanan, K. Taht, R. Balasubramonian, E. Giacomin, and P.-E. Gaillardon, "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, W. Zhang, and P. Li, "Parallel time batching: Systolic-array acceleration of sparse spiking neural computation," in *HPCA*, 2022.
[13] H. W. Lui and E. Neftci, "Hessian aware quantization of spiking neural networks," in *ICONS*, 2021.
[14] C. J. Schaefer and S. Joshi, "Quantizing spiking neural networks with integers," *ICONS*, 2020.
[15] N. Rathi, P. Panda, and K. Roy, "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 and M. Shafique, "Q-spinn: A framework for quantizing spiking neural networks," in *IJCNN*, 2021.
[18] A. R. Voelker, D. Rasmussen, and C. Eliasmith, "A spike in performance: Training hybrid-spiking neural networks with quantized activation functions," *arXiv:2002.03553*, 2020.
[19] C. Li, L. Ma, and S. Furber, "Quantization framework for fast spiking neural networks," *Frontiers in Neuroscience*, 2022.
[20] R. Yin, A. Moitra, A. Bhattacharjee, Y. Kim, and P. Panda, "Sata: Sparsity-aware training accelerator for spiking neural networks," *TCAD*, 2022.
[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.
[22] K. He, X. Zhang, S. Ren, and J. Sun, "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. Li, H. Liu, X. Ji, G. Li, and L. Shi, "Cifar10-dvs: an event-stream dataset for object classification," *Frontiers in neuroscience*, 2017.
[25] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, 2009.
[26] Y. Kim and P. Panda, "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.