A APPENDIX

A.1 QUANTIZATIONS FORMATS

In this paper, we aim to reduce the General matrix multiplication (GEMM) to 4 bit. The activation and weights are quantized to standard INT4, while the neural gradients to FP4 with LUQ. In the forward path (Eq. (1)), we use standard INT quantization (Equation 3 in Nagel et al. (2021)):

$$z_l = \sum Q_{INT4}(W_l) Q_{INT4}(a_l) = \sum \alpha_{W_l} W_l^{INT4} \alpha_{a_l} a_l^{INT4} = \alpha_{W_l} \alpha_{a_l} \sum W_l^{INT4} a_l^{INT4} \,,$$

where α is the INT scale factor and defined as: $\alpha = \frac{\Delta}{2^b - 1}$ where Δ is the quantized range and b is the quantization bitwidth. In this standard INT quantization all MAC operations are in low precision and only the final multiplication with the scale requires high precision. For the backward (Eq. (2)) and update (Eq. (3)) GEMMs we use exactly the same scheme: again the scale (α) is in high precision but since it is the same for all the tensor, all MAC operation can be done in low precision. The only difference is that we require a MAC unit that performs INT4-FP4 operations. This requirement is the same as required in Ultra-low (Sun et al. (2020)) and as they showed, the support of such INT4-FP4 MAC unit is simple and requires 55% of the area of standard FP16 Floating-point-unit, while providing a 4x throughput. In Fig. 3 we show all these details.

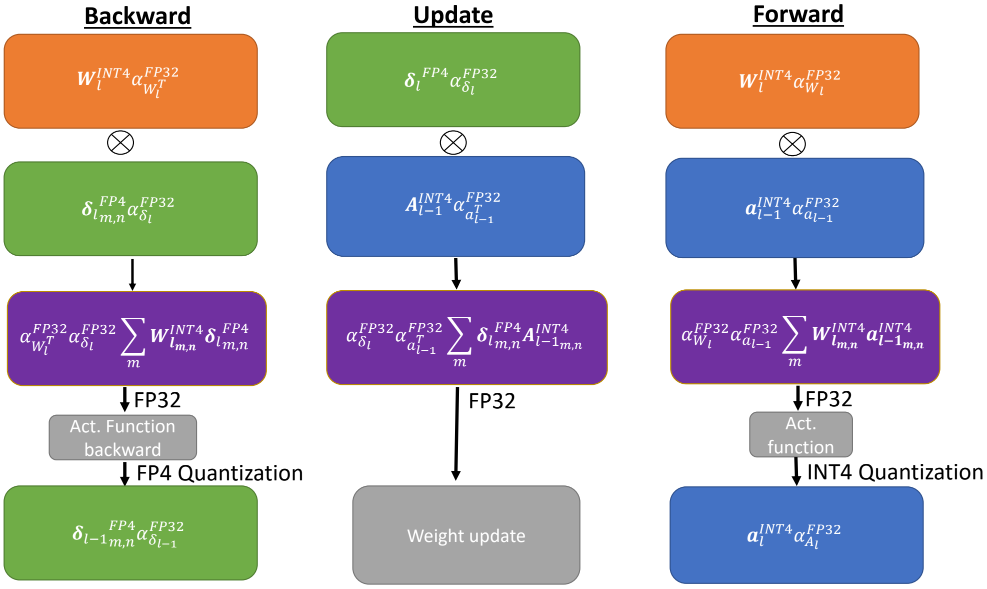

Backward
Update
Forward
$W_l^{INT4} \alpha_{W_l^T}^{FP32}$
$\delta_l^{FP4} \alpha_{\delta_l}^{FP32}$
$W_l^{INT4} \alpha_{W_l}^{FP32}$
$\otimes$
$\delta_{l_{m,n}}^{FP4} \alpha_{\delta_l}^{FP32}$
$A_{l-1}^{INT4} \alpha_{a_{l-1}^T}^{FP32}$
$a_{l-1}^{INT4} \alpha_{a_{l-1}}^{FP32}$
$\alpha_{W_l^T}^{FP32} \alpha_{\delta_l}^{FP32} \sum_m W_{l_{m,n}}^{INT4} \delta_{l_{m,n}}^{FP4}$
$\alpha_{\delta_l}^{FP32} \alpha_{a_{l-1}^T}^{FP32} \sum_m \delta_{l_{m,n}}^{FP4} A_{l-1_{m,n}}^{INT4}$
$\alpha_{W_l}^{FP32} \alpha_{a_{l-1}}^{FP32} \sum_m W_{l_{m,n}}^{INT4} a_{l-1_{m,n}}^{INT4}$
FP32
Act. Function backward
FP4 Quantization
FP32
FP32
Act. function
INT4 Quantization
$\delta_{l-1_{m,n}}^{FP4} \alpha_{\delta_{l-1}}^{FP32}$
Weight update
$a_l^{INT4} \alpha_{A_l}^{FP32}$

Figure 3: Summary of the different formats used in each of the three GEMM in our 4 training regime. Boldface represent matrix and vectors. Different colors represent different elements of each layer (activations, weights, neural gradients, etc.)

A.2 COMPARISON OF MEAN-SQUARE-ERROR - FULL DERIVATIVES

Given that we want to quantize x in a bin with a lower limit $l(x)$ and an upper limit $u(x)$, stochastic rounding can be stated as follows:

$$\mathrm{SR}(x) = \begin{cases} l(x), & w.p. \quad p(x) = 1 - \frac{x - l(x)}{u(x) - l(x)} \\ u(x), & w.p. \quad 1 - p(x) = \frac{x - l(x)}{u(x) - l(x)} \end{cases} . \tag{19}$$

The expected rounding value is given by

$$E[\mathrm{SR}(x)] = l(x) \cdot p(x) + u(x) \cdot (1 - p(x)) = x \,, \tag{20}$$

where here and below the expectation is over the randomness of SR (i.e., x is a deterministic constant).

Stochastic rounding is an unbiased approximation of x, since it has zero bias:

$$\mathrm{Bias}[\mathrm{SR}(x)] = E[\mathrm{SR}(x) - x] = E[\mathrm{SR}(x)] - x = 0\,. \tag{21}$$

However, stochastic rounding has a variance, given by

$$\begin{aligned}\mathrm{Var}[\mathrm{SR}(x)] &= (l(x) - E[\mathrm{SR}(x)])^2 \cdot p(x) + (u(x) - E[\mathrm{SR}(x)])^2 \cdot (1 - p(x)) \\ &= (x - l(x)) \cdot (u(x) - x)\ ,\end{aligned} \tag{22}$$

where the last transition follows from substituting the terms $E[\mathrm{SR}(x)])$, and $p(x)$ into Eq. (22).

We turn to consider the round-to-nearest method (RDN). The bias of RDN is given by

$$\mathrm{Bias}[\mathrm{RDN}(x)] = \min\left(x - l(x), u(x) - x\right). \tag{23}$$

Since RDN is a deterministic method, it is evident that the variance is 0 i.e.,

$$\mathrm{Var}[\mathrm{RDN}(x)] = 0\,. \tag{24}$$

Finally, for every value x and a rounding method $R(x)$, the mean-square-error (MSE) can be written as the sum of the rounding variance and the squared rounding bias,

$$\mathrm{MSE}[R(x)] = E[R(x) - x]^2 = \mathrm{Var}[\mathrm{R}(x)] + \mathrm{Bias}^2[\mathrm{R}(x)]\,. \tag{25}$$

Therefore, we have the following MSE distortion when using round-to-nearest and stochastic rounding:

$$\mathrm{MSE} = \begin{cases}\left[\min\left(x - l(x), u(x) - x\right)\right]^2 & RDN(x) \\ (x - l(x)) \cdot (u(x) - x) & SR(x)\end{cases}\ . \tag{26}$$

A.3 WHEN IS IT IMPORTANT TO USE UNBIASED QUANTIZATION - FULL DERIVATIVES

Denote W_l as the weights between layer $l-1$ and l, C as the cost function, and f_l as the activation function at layer l. Given an input–output pair (x, y), the loss is:

$$C\left(y, f_L\left(W_L f_{L-1}\left(W_{L-1} \cdots f_2\left(W_2 f_1\left(W_1 x\right)\right) \cdots\right)\right)\right)\ . \tag{27}$$

Backward pass. Let z^l be the weighted input (pre-activation) of layer l and denote the output (activation) of layer l by a_l. The derivative of the loss in terms of the inputs is given by the chain rule:

$$\delta_l \triangleq \frac{dC}{dz_l} = \frac{da_l}{dz_l} \cdots \frac{dz_{L-1}}{da_{L-2}} \cdot \frac{da_{L-1}}{dz_{L-1}} \cdot \frac{dz_L}{da_{L-1}} \cdot \frac{da_L}{dz_L} \cdot \frac{dC}{da_L}\,. \tag{28}$$

Therefore, $\delta_L \triangleq \frac{da_L}{dz_L} \cdot \frac{dC}{da_L}$, and we can write recursively the backprop rule $\forall l < L$:

$$\delta_l \triangleq \frac{da_l}{dz_l}\frac{dz_{l+1}}{da_l} \cdot \delta_{l+1}\,. \tag{29}$$

In its quantized version, $\delta_L^q = Q\left(\delta_L\right)$, and Eq. (29) has the following form (with ReLU activations):

$$\delta_l^q \triangleq Q\left(\frac{da_l}{dz_l}\frac{dz_{l+1}}{da_l}\delta_{l+1}^q\right)\ . \tag{30}$$

Assuming $Q(x)$ is an unbiased stochastic quantizer with $E[Q(x)] = x$, we next show the quantized backprop δ_l^q is an unbiased approximation of backprop:

$$\begin{aligned}E\delta_l^q = EQ\left(\frac{da_l}{dz_l}\frac{dz_{l+1}}{da_l}\delta_{l+1}^q\right) &\overset{(1)}{=} E\left[E\left[Q\left(\frac{da_l}{dz_l}\frac{dz_{l+1}}{da_l}\delta_{l+1}^q\right)|\delta_{l+1}^q\right]\right] \\ \overset{(2)}{=} E\left[\frac{da_l}{dz_l}\frac{dz_{l+1}}{da_l}\delta_{l+1}^q\right] &\overset{(3)}{=} \frac{da_l}{dz_l}\frac{dz_{l+1}}{da_l}E\delta_{l+1}^q \overset{(4)}{=} \frac{da_l}{dz_l}\frac{dz_{l+1}}{da_l}\delta_l\,,\end{aligned} \tag{31}$$

where in (1) we used the law of total expectation, in (2) we used $E[Q(x)] = x$, in (3) we used the *linearity* of back-propagation, and in (4) we assumed by induction that $E\left[\delta_{l+1}\right] = \delta_{l+1}^q$, which holds initially: $E\delta_L^q = EQ\left(\delta_L\right) = \delta_L$.

Finally, the gradient of the weights in layer l is $\nabla_{W_l} C = \delta_l a_{l-1}^\top$ and its quantized form is $\nabla_{W_l} C_q = \delta_l^q a_{l-1}^\top$. Therefore, the update $\nabla_{W_l} C_q$ is an unbiased estimator of $\nabla_{W_l} C$:

$$E\left[\nabla_{W_l} C_q\right] = E\left[\delta_l^q a_{l-1}^\top\right] = E\left[\delta_l^q\right] a_{l-1}^\top = \delta_l a_{l-1}^\top = \nabla_{W_l} C\,. \tag{32}$$

A.4 EXPERIMENTAL DETAILS

In all our experiments we use the most common approach (Banner et al., 2018; Choi et al., 2018b) for quantization where a high precision of the weights is kept and quantized on-the-fly. The updates are done in full precision. In all our experiments we use 8 GPU GeForce Titan Xp or GeForce RTX 2080 Ti or Ampere A40.

ResNet / ResNext We run the models ResNet-18, ResNet-50 and ResNext-50 from torchvision. We use the standard pre-processing of ImageNet ILSVRC2012 dataset. We train for 90 epochs, use an initial learning rate of 0.1 with a 0.1 decay at epochs 30,60,80. We use standard SGD with momentum of 0.9 and weight decay of 1e-4. The minibatch size used is 256. Following the DNNs quantization conventions (Banner et al., 2018; Nahshan et al., 2019; Choi et al., 2018b) we kept the first and last layer (FC) at higher precision. Additionally, similar to Sun et al. (2020) we adopt the full precision at the shortcut which constitutes only a small amount of the computations ($\sim 1\%$). We totally The ”underflow threshold” in LUQ is updated in every bwd pass as part of the quantization of the neural gradients. In all experiments, the BN and pooling are calculated in high-precision. The hindsight momentum is $\eta = 0.1$ and in the FNT experiments we use $\text{lr}_{base} = 1e - 3$.

MobileNet V2 We run Mobilenet V2 model from torchvision. We use the standard pre-processing of ImageNet ILSVRC2012 dataset. We train for 150 epochs, use an initial learning rate of 0.05 with a cosine learning scheduler. We use standard SGD with momentum of 0.9 and weight decay of 4e-5. The minibatch size used is 256. Following the DNNs quantization conventions (Banner et al., 2018; Nahshan et al., 2019; Choi et al., 2018b) we kept the first and last layer (FC) at higher precision. Additionally, similar to Sun et al. (2020) we adopt the full precision at the depthwise layer which constitutes only a small amount of the computations ($\sim 3\%$). The ”underflow threshold” in LUQ is updated in every bwd pass as part of the quantization of the neural gradients. In all experiments, the BN and pooling are calculated in high-precision.

Transformer We run the Transformer-base model based on the Fairseq implementation on the WMT 14 En-De translation task. We use the standard hyperparameters of Fairseq including Adam optimizer. We implement LUQ over all attention and feed forward layers.

A.5 ADDITIONAL EXPERIMENTS

A.5.1 STOCHASTIC ROUNDING OVERHEAD

Efficient HW implementation of SR can be found in many accelerators that were built specifically for deep learning (Habana-Intel Hab, Graphcore Gra, Tesla Tes). In order to show the throughput overhead of stochastic rounding, when it is not supported in HW as in GPU, we measured the time of our stochastic quantizer in comparison to a deterministic quantizer (round-to-nearest) for different sizes of random tensor. The quantizer was implemented as a CUDA kernel and the experiment run on 1 A40 GPU. For each tensor size, we repeat the experiment 100 times and average the results. We see that the implementation of SR in software with our non-optimized kernel has a small overhead. This can be an option when it is not supported in HW.

Tensor size	Stochastic rounding [micro sec]	Round-to-nearest [micro sec]
10^3	2.64	2.55
10^4	2.64	2.6
10^5	3.94	3.89
10^6	4.07	4.01
10^7	6.89	6.78
10^8	9.83	9.77

A.5.2 STOCHASTIC ROUNDING AMORTIZATION

As explained in Appendix A.5.1, usually the overhead of the stochastic rounding is typically negligible in comparison to other operations in neural network training. However, to reduce even more this overhead, is it possible to re-use the random samples. In Fig. 4 we show the effect of such re-using does not change the network accuracy.

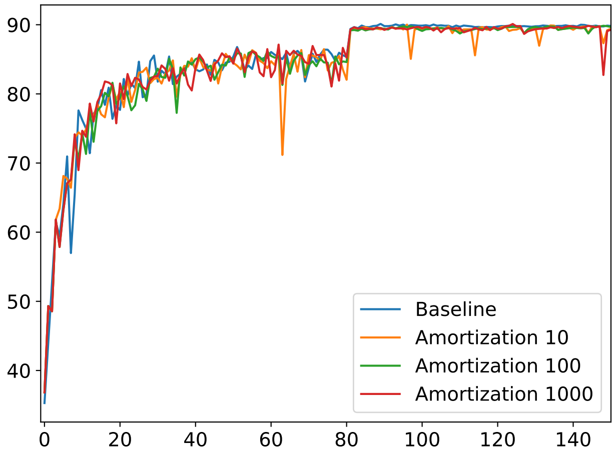


Figure 4: ResNet18 top-1 validation in Cifar10 dataset, with 4-bit quantization of the neural gradients using stochastic-rounding. The amortization is the numbers of iteration that we re-use the random samples.

A.5.3 SMP EXPERIMENT

In Fig. 5 we show the effect of the different number of samples (SMP) on 2-bit quantization of ResNet18 Cifar100 dataset. There, we achieve with 16 samples accuracy similar to a full-precision network. This demonstrates that the variance is the only remaining issue in neural gradient quantization using LUQ, and that the proposed averaging method can erase this variance gap, with some overhead.

A.5.4 SMP OVERHEAD

The SMP method (Section 5.1) has a power overhead of $\sim \frac{1}{3}$ of the number of additional samples since it influences only the update GEMM. In Fig. 6 we compare LUQ with one additional sample which has $\sim 33\%$ power overhead with regular LUQ with additional $\sim 33\%$ epochs. The learning rate scheduler was expanded respectively. We can notice that, even though both methods have a similar overhead, the variance reduction achieved with SMP is more important for network accuracy than increasing the training time.

A.5.5 DATA MOVEMENT REDUCTION EFFECT

LUQ requires the measurement of the maximum to choose the underflow threshold α (Section 4). This measurement can create a data movement bottleneck. In order to avoid it, we combine in LUQ the proposed maximum estimation of Hindsight Fournarakis & Nagel (2021), which uses previous iterations statistics. In Fig. 7 we compare the measured maximum and the Hindsight estimation, showing they can have similar values. Moreover in Table 4 we show that the effect of the Hindsight estimation on the network accuracy is negligible while completely eliminating the data movement bottleneck. In Table 5 we extend Table 3 and show the effect of applying the proposed FNT method (Section 5.2) on the combination of LUQ with Hindisght.

A.6 COMPARISON TO SUN ET AL. (2020)

Floating point radix conversion requires an explicit multiplication and may require additional non-standard hardware support (Kupriianova et al., 2013). Specifically, Sun et al. (2020) requires a conversion from the radix-2 FP32 to a radix-4 FP4 of the neural gradients. They show this conversion requires multiplication by the constant 1.6.

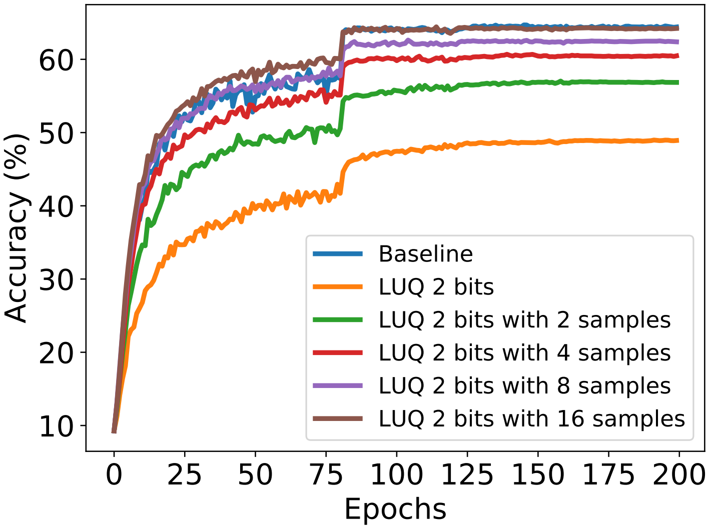


Figure 5: ResNet18 top-1 validation accuracy in CIFAR100 with quantization of the neural gradients to 2-bit (FP2 - [1,1,0] format) using different samples numbers to reduce the variance. Notice that 16 samples completely close the gap to the baseline.

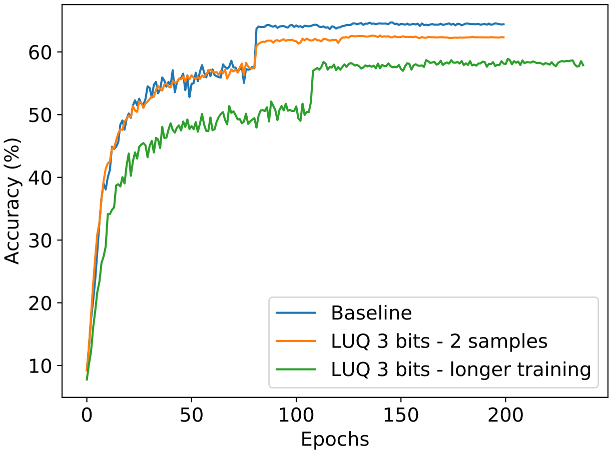
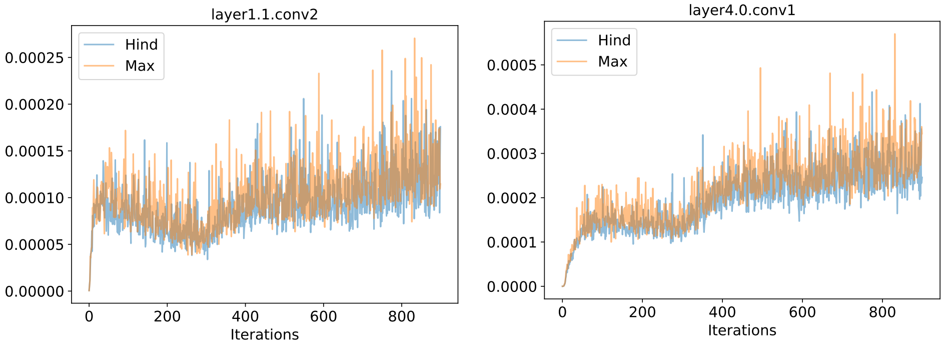


Figure 6: Comparison of ResNet-18 3 bit training on Cifar100 dataset of LUQ with 2 samples with longer training of regular LUQ. Both methods have similar overhead, but the SMP method leads to better accuracy.

Notice that it is not possible to convert between radix floating point formats by a fixed shift. For example, suppose we first convert the radix-2 FP32 to radix-2 FP4 and then shift the exponent (where the shift is equivalent to multiplication by 2). This would lead to an incorrect result, as we show with a simple example: let us assume radix-2 quantization with bins 1,2,4,8 and radix-4 quantizations with bins 1,4,16,64. For the number 4.5, if we quantize it first to radix-2 we get the quantized number 4, then we multiply it by 2 we get 8. In contrast, radix-4 quantization should give the result 4.

In contrast, our proposed method, LUQ, uses the standard radix-2 format and does not require non-standard conversions. The use of standard hardware increases the benefit of the low bits quantization.



(a) (b)

Figure 7: Measured maximum and hindsight Fournarakis & Nagel (2021) maximum estimation in two different layers of ResNet18 ImageNet dataset. As you can notice, the hindsight estimation is close to the measured maximum and reduce completely the data movement bottleneck

Table 4: Effect of applying Hindsight Fournarakis & Nagel (2021) maximum estimation on the network accuracy. SMP refers to doing two samples of the SR quantization of neural gradient in order to reduce the variance as explained in Section 5.1 similar to Table 2

Model	LUQ	LUQ + SMP	LUQ + Hindsight	LUQ + Hindsight + SMP
ResNet18	69.09	69.24	69.12	69.25
ResNet50	75.42	75.63	75.4	75.6
MobileNet-V2	69.55	69.7	69.59	69.68
ResNext50	76.02	76.12	75.91	76.2
Transformer-base	27.17	27.25	27.17	27.23
BERT fine-tune	85.75	85.9	85.77	85.89

Table 5: Effect of the proposed FNT method (Section 5.2) on LUQ and on LUQ + Hindisght (Fournarakis & Nagel, 2021) similar to Table 3. FNT 1/2/3 refers to the number of finetuning epochs, while ”No FNT” refers to the basic method + SMP as presented in Table 4

Model	Baseline	No FNT	+FNT 1	+FNT 2	+FNT 3
ResNet-18 LUQ	69.7 %	69.24 %	69.7 %	-	-
ResNet-18 LUQ+Hind	69.7 %	69.25 %	69.7 %	-	-
ResNet-50 LUQ	76.5 %	75. 63%	75.89 %	76 %	76.18 %
ResNet-50 LUQ + Hind	76.5 %	75. 6%	75.84 %	76.03 %	76.25 %
MobileNet-V2 LUQ	71.9 %	69.7 %	70.1 %	70.3 %	70.3 %
MobileNet-V2 LUQ + Hind	71.9 %	69.68 %	70.1 %	70.3 %	70.3 %
ResNext-50 LUQ	77.6 %	76.12%	76.25 %	76.33 %	76.7 %
ResNext-50 LUQ + Hind	77.6 %	76.2%	76.3 %	76.43 %	76.65 %

Table 6: ResNet-50 accuracy with ImageNet dataset while using quantization on different parts of the network. The forward phase is quantized to INT4 format with SAWB (Choi et al., 2018a) while the backward phase is quantized with the proposed LUQ. As expected, the quantization of the backward phase makes more degradation to the network accuracy.

Forward	Backward	Accuracy
FP32	FP32	76.5 %
INT4	FP32	76.35 %
FP32	FP4	75.6 %
INT4	FP4	75.4 %

A.7 POWER-OF-TWO LUQ

Recall the quantization bins in LUQ are $2^n\alpha$ ($n \in \{0, 1, .., b-1\}$), where α is a real number defined as (with $b = 3$ for FP4):

$$\alpha = \frac{\max(|x|)}{2^{2^{b-1}}}\ .$$

In order to reduce the computational resources in the quantization, we can use only power-of-two values for α. This allows us to convert all bins to power-to-two values. This enables using a cheap shift operation instead of the more expensive multiplication with α we do during the quantization process (Eq. (12)).

Specifically, we suggest power-of-two LUQ (LUQ$_{\text{PW2}}$), which uses the ceiling power-of-two value of the maximum in the α calculation, i.e.:

$$\alpha_{\text{PW2}} = \frac{2^{\lceil \log_2 \max(|x|) \rceil}}{2^{2^{b-1}}}\ .$$

The choice of the ceiling (instead of round-to-nearest) is to avoid clipping of the maximum value which will create a bias and affect the accuracy.

Standard FP quantization, with E exponent bits and M mantissa has a dynamic range of $[2^{1-q}, 2^{2^E-2-q}(2-2^{-M})]$ where q is the exponent bias. Usually, the exponent bias is fixed as $q = 2^{E-1} - 1$. However, in modern deep learning accelerator Tes the FP quantizers have the ability to change the exponent bias. With this ability, during the use of LUQ$_{\text{PW2}}$, we can define the exponent bias for a tensor x to be $q = 2^3 - 2 - \lceil \log_2(\max(|x|)) \rceil$ (for $E = 3$ and $M = 0$). This bias allows us to completely avoid any shifting operations, since it is defined as part of the quantizer. Thus we obtain an even more significant reduction in the computational resources required to use LUQ.

In Table 7 we show results of the proposed LUQ$_{\text{PW2}}$ achieving a small degradation in comparison to standard LUQ and reducing or completely avoiding (depends on the accelerator) the computational cost of the scaling operator. Moreover in Table 8 we combine LUQ$_{\text{PW2}}$ with Hindsight Fournarakis & Nagel (2021) to additionally reduce the data movement, showing better results than Sun et al. (2020) with their non hardware friendly method.

Table 7: Comparison of Ultra-low (Sun et al., 2020), LUQ and LUQ$_{\text{PW2}}$ on various models and datasets. As can be seen, the proposed LUQ$_{\text{PW2}}$ avoids the scaling operations and achieved a small degradation in comparison to standard LUQ.

Model	Baseline	Ultra-low	LUQ	LUQ$_{\text{PW2}}$
ResNet-18	69.7 %	68.27 %	69.09%	69 %
ResNet-50	76.5%	74.01 %	75.42 %	75.15 %
MobileNet-V2	71.9 %	68.85 %	69.55 %	69.2 %
ResNext50	77.6 %	N/A	76.02 %	75.3 %
Transformer-base	27.5 (BLEU)	25.4	27.17	26.86
BERT fine-tune	87.03 (F1)	N/A	85.75	85.29

Table 8: Combination of the proposed LUQ$_{\text{PW2}}$ and Hindsight Fournarakis & Nagel (2021) on ResNet18 and ResNet50 ImageNet dataset. As can be seen, the Hindsight method has a small effect on the accuracy and allow to reduce the data movement, which can be critical in some cases.

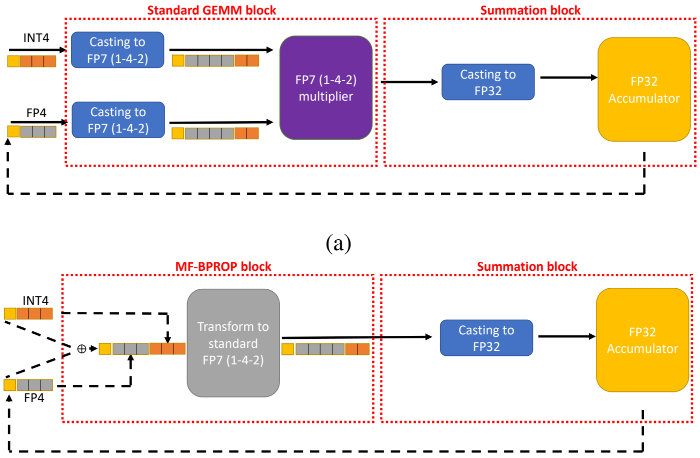
Model	Baseline	Ultra-low	LUQ$_{\text{PW2}}$	LUQ$_{\text{PW2}}$ + Hindsight
ResNet-18	69.7 %	68.27 %	68.7 %	68.88 %
ResNet-50	76.5 %	74.01%	75.15 %	74.83 %

A.8 MF-BPROP: MULTIPLICATION FREE BACKPROPAGATION

The main problem of using different datatypes for the weights, activations and neural gradients is the need to cast them to a common data type before the multiplication during the backward (Eq. (2))

and update(Eq. (3)) phases. During the backward and update phases, in each layer l there are two GEMMs between different datatypes:
Regularly, to calculate these GEMMs there is a need to cast both data types to a common data type (in our case, FP7 [1,4,2]), then do the GEMM and finally, the results are usually accumulated in a wide accumulator (Fig. 8a). This casting cost is not negligible. For example, casting INT4 to FP7 consumes $\sim 15\%$ of the area of an FP7 multiplier.

In our case, we are dealing with a special case where we do a GEMM between a number without mantissa (neural gradient) and a number without exponent (weights and activations), since INT4 is almost equivalent to FP4 with format [1,0,3]. We suggest transforming the standard GEMM block (Fig. 8a) to Multiplication Free BackPROP (MF-BPROP) block which contains only a transformation to standard FP7 format (see Fig. 8b) and a simple XOR operation. More details on this transformation appear in Appendix A.8.1. In our analysis (Appendix A.8.2) we show the MF-BPROP block reduces the area of the standard GEMM block by $5\times$. Since the FP32 accumulator is still the most expensive block when training with a few bits, we reduce the total area in our experiments by $\sim 8\%$. However, as previously showen (Wang et al., 2018) 16-bits accumulators work well with 8-bit training, so it is reasonable to think, it should work also with 4-bit training. In this case, the analysis (Appendix A.8.2) shows that the suggested MF-BPROP block reduces the total area by $\sim 22\%$.



(a)

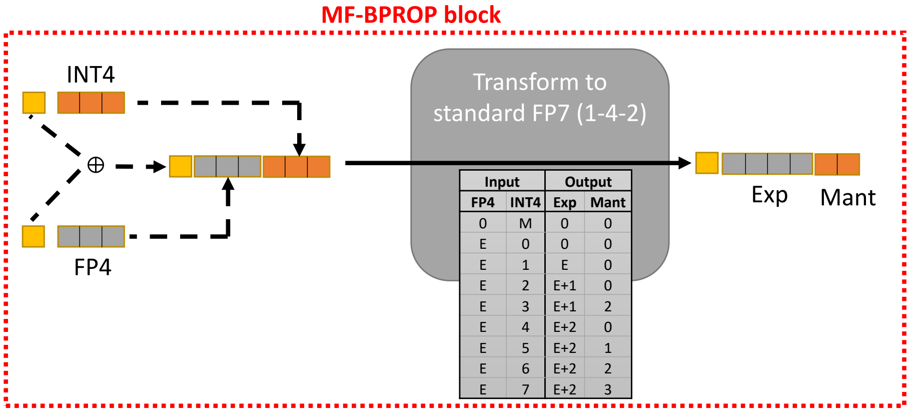
(b)

Figure 8: **(a):** Standard MAC block illustration containing the two main blocks - one for GEMM and second for accumulator. The GEMM block for hybrid datatype as in our case (FP4 and INT4) requires casting to a common datatype before being inserted into the multiplier. **(b):** The suggested MAC block, which replaces the multiplier with the proposed MF-BPROP. Instead of doing an expensive casting followed by multiplication, we propose to make only a simple XOR and a transformation (Appendix A.8.1) reducing the GEMM area by 5x (Appendix A.8.2).

A.8.1 TRANSFORM TO STANDARD FP7

We suggest a method to avoid the use of an expensive GEMM block between the INT4 (activation or weights) and FP4 (neural gradient). It includes 2 main elements: The first is a simple xor operation between the sign of the two numbers and the second is a transform block to standard FP7 format. In Fig. 9 we present an illustration of the proposed method. The transformation can be explained with a simple example: for simplicity, we avoid the sign which requires only xor operation. The input arguments are 3 (011 bits representation in INT4 format) and 4 (011 bits representation in FP4 1-3-0 format). The concatenation brings to the bits 011 011. Then looking at the table in the input column where the M=3 (since the INT4 argument = 3) and get the results in FP7 format of 0100 10 (= E+1 2) which is 12 in FP7 (1-4-2) as the expected multiplication result.

In the next section, we analyze the area of the suggested block in comparison to the standard GEMM block, showing a $5\times$ area reduction.



Input		Output	
FP4	INT4	Exp	Mant
0	M	0	0
E	0	0	0
E	1	E	0
E	2	E+1	0
E	3	E+1	2
E	4	E+2	0
E	5	E+2	1
E	6	E+2	2
E	7	E+2	3

Figure 9: Illustration of MF-BPROP block which replaces a standard multiplication. It includes: (1) a simple xor operation between the sign. (2) A transform to standard FP7 format. We present the table to make this transform - E and M represent the bits of the FP4 and INT4 respectively without the sign. Exp and Mant are the bits of the output exponent (4-bit) and mantissa (2-bit) of the output in FP7 format.

A.8.2 BACKPROPAGATION WITHOUT MULTIPLCATION ANALYSIS

In this section, we show a rough estimation of the logical area of the proposed MF-BPROP block which avoids multiplication and compares it with the standard multiplier. In hardware design, the logical area can be a good proxy for power consumption (Iman & Pedram, 1997). Our estimation doesn't include synthesis optimization. In Table 9 we show the estimation of the number of gates of a standard multiplier, getting 264 logical gates while the proposed MF-BPROP block has an estimation of 49 gates (Table 10) achieving a $\sim 5\times$ area reduction. For fair comparison we remark that in the proposed scheme the FP32 accumulator is the most expensive block with an estimation of 2453 gates, however we believe it can be reduced to a narrow accumulator such as FP16 (As previously shown in Wang et al. (2018) which have an estimated area of 731 gates. In that case, we reduce the total are by $\sim 22\%$.

Table 9: Rough estimation of the number of logical gates for a standard GEMM block which contains two blocks: a casting to FP7 and a FP7 multiplier.

Block	Operation	# Gates
Casting to FP7	Exponent 3:1 mux	12
	Mantissa 4:1 mux	18
FP7 [1,4,2] multiplier	Mantissa multiplier	99
	Exponent adder	37
	Sign xor	1
	Mantissa normalization	48
	Rounding adder	12
	Fix exponent	37
Total		**264**

Table 10: Rough estimation of the number of logical gates for the proposed MF-BPROP block.

Block	Operation	# Gates
MF-BPROP	Exponent adder	30
	Mantissa 4:1 mux	18
	Sign xor	1
Total		**49**