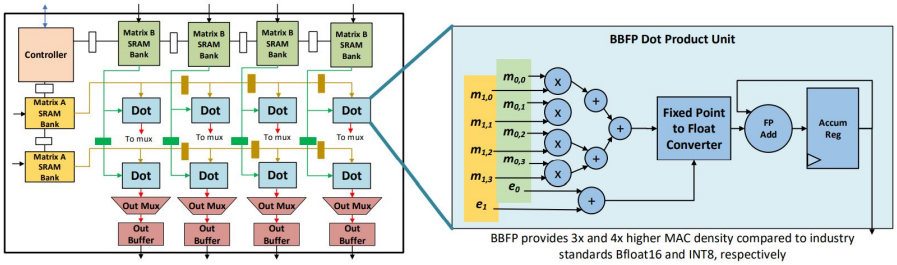
We thank the reviewers for their appreciation of our work and helpful feedback. Here, we address open questions.

[Reviewer 7] Reproducibility of hardware: Following figure shows a systolic tensor core architecture containing multiple BBFP dot product units. Each dot product unit has a significantly lower circuit footprint compared to conventional float due to the shared exponent. The math per bounding-box is mostly performed in fixed-point format and the cost of dynamic scaling is amortized over n (block size). We'll amend the camera-ready with more details.



[Reviewer 7 and 1] Conversion to BBFP: the axis along which exponent is shared is always the inner dimensions of a matrix multiply: for $\mathbf{A} \times \mathbf{B}$ we have shared exponents along the rows of $\mathbf{A}$ and the columns of $\mathbf{B}$. If the inner dimension is not dividable by the bounding box size, we pad the last bounding box with zero.

[Reviewer 3 and 2] Comparison with FlexPoint, INT8, INT4, and AdaptivFloat: As discussed in the paper, leveraging shared exponent to lower hardware cost is not a new idea. However, there is a very large search space of designs across bit-width, exponent selection policy, block granularity, bounding box policy—coupled with hardware implementation—that makes BBFP non-obvious. In this paper, we showed how using a balanced fine-grained approach can provide a robust recipe that works across various models. Due to the low user-friction of BBFP format and its high performance, this datatype has been adopted by different teams and deployed in large-scale production. FlexPoint coarse-grained approach, however, results in significant accuracy drops (in presence of outliers) and incur a high-friction pass to recover accuracy. We evaluated the accuracy versus MAC density trade-off for different benchmarks in Figures 1 and 5. This comparison includes industry-standard datatypes (bfloat16, INT8, and INT4). Different datatypes' circuitry is compared in Table 1. We'll append Table 1 to include float8, float4, and posits in the revision. Overall, float8 has $1.57\times$ and $3.46\times$ more area overhead compared to BBFP16 and BBFP12, respectively. As for comparison with AdaptiveFloat, in ResNet50-ImageNet with an average of 4 bits (sign plus mantissa), BBFP preserves 96.7% of the baseline accuracy whereas AdaptivFloat preserves 95.2%. Also, BBFP uses a uniform bit-width for all layers whereas AdaptivFloat adjusts the bit-width per layer, making it more complicated on hardware. Finally, AdaptivFloat reports up to $1.14\times$ area improvement compared to INT8 whereas BBFP has a $4\times$ lower overhead.

[Reviewer 3 and 2] Fine-tuning process and calibration: Quantization involves discretization processes such as rounding and truncation that result in null gradients. The idea of straight-through estimator is to replace the gradient of those operators with identity matrix. We'll elaborate more in the revision. Integer-based inference requires calibration due to their fixed dynamic range—tensors must be scaled to the proper range to be represented. BBFP does not require this type of calibration as it already has a scaling exponent. As for bounding-box selection, all experiments in section 4 including Tables 3, 4, and Figure 5 have been performed with a fixed bounding box size of 16 (no calibration involved).

[Reviewer 2] Elaboration on Figure 1: BBFP12 has 4-bit sign-magnitude mantissas and an 8-bit shared exponent. BBFP12 dot-product circuitry is comparable to INT4. Unlike BBFP, INT format follows a two's complement representation. Two's complement MAC costs more in area/energy compared to a sign-magnitude MAC of the same bit-width. We'll make the notation more explicit in the revision. MAC energy/area cost are measured based on a topographical synthesis of these MAC units on TSMC 16nm FF+. Power simulations are based on an input toggle rate of 50% with 50% static probability @1GHz. Figure 1 accuracy numbers are for Resnet50-ImageNet.

[Reviewer 1] Non-matmul operations and support for strided and atrous convolution: BBFP is designed to improve dot product performance. All vector operations such as sigmoid, activation functions, point-wise addition are performed in-situ on HW in float16 (with conversions to BBFP supported automatically on HW). Any operation that can be broken down into dot products (including strided/dilated convs) is supported by BBFP. Scatter/gather is one way of reshaping the input data for strided/dilated convs (SRAM address striding and crossbar are other alternatives).

[Reviewer 1] Zero representation: Zero is represented by having all mantissa bits being 0 for a given value (shared exponent can be any value). BBFP mantissas do not have an implicit leading bit and all mantissa bits are explicitly represented. BBFP does not have a representation for NaN/Inf, but this does not impact DNN inference accuracy.

[Reviewer 7 and 2] KL Divergence and QNSR: The KL divergence from BBFP to float32 is computed after computing the pertinent normalized histogram of values in each encoding format. KL divergence is defined as $KL(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$. We will replace QNSR with QSNR in the revised paper to avoid negative values.