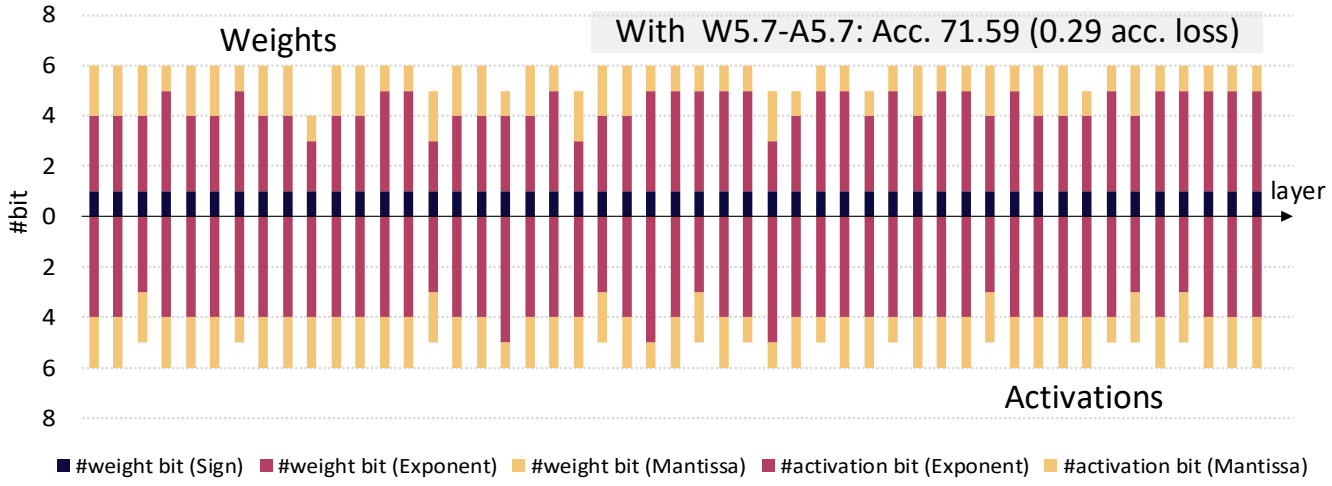
Appendix A. Experimental Setup

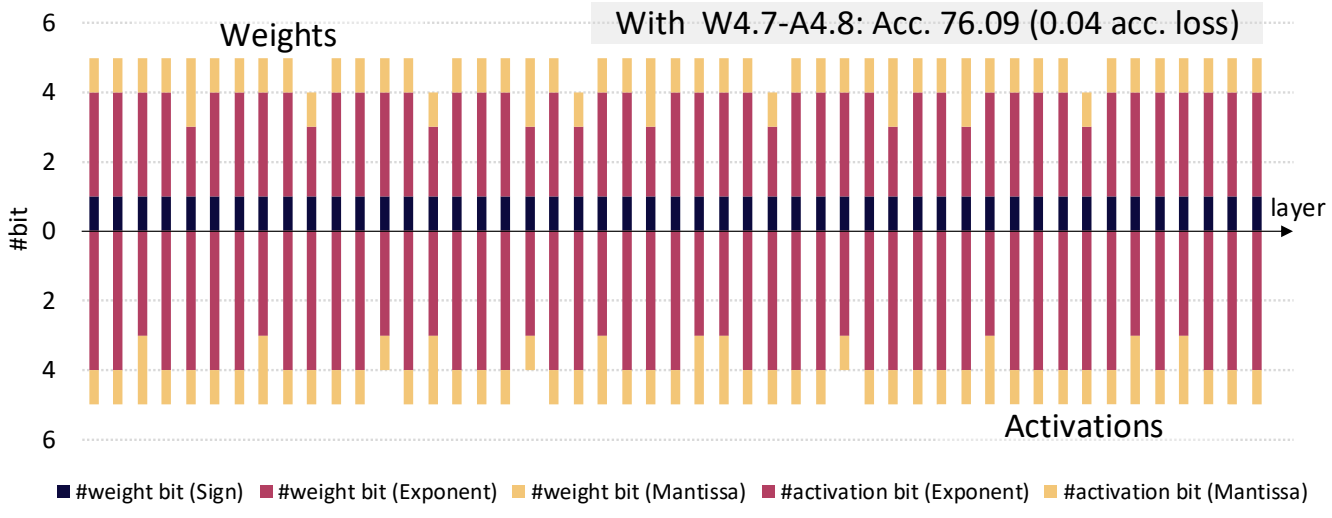
All the experiments were performed in the PyTorch framework using a 4-way NVIDIA RTX-2080ti GPUs. For ImageNet, the experiments are conducted on ResNet-50 [10] and MobileNet-v2 [26], which covers both the popular normal-size model and the state-of-the-art compact model. We adopt the identical data augmentation configuration as ResNet series [10]. Moreover, all the reported classification accuracy on the validation dataset is the single-crop result. The batch size used for inference is 128. We also want to highlight that both the first and last layers are quantized in this work. Besides, thanks to the representation superiority of our proposed AFP format, even with low overall bit-width ($n_{\text{man}} + n_{\text{exp}} + 1$), negligible accuracy degradation can be achieved without time-consuming retraining.

Appendix B. Layer-wise Bit-width

The bit-with for each layer of the reported ResNet-50 and MobileNet-V2 (in Table 2) models encoded with AFP can be found in Fig.9.Above the abscissa (Fig.9) is the bit-width of each layer of the weights. The bit width is composed of three parts: sign bit, exponent part, and mantissa part. Meanwhile, below the abscissa (Fig.9) is the bit-width distribution of activations for each layer, composed of the exponent part and mantissa part. Because, after the ReLu activation function, the activations are all greater than 0. For the activations, our AFP allocates more bit-width for the exponent part since the activations always have a greater dynamic range.



(a) Layer-wise bit-width of AFP encoded MobileNet-v2



(b) Layer-wise bit-width of AFP encoded ResNet-50

Figure 9: On ImageNet dataset, the layer-wise bit-width of AFP encoded ResNet-50 and MobileNet-v2 model.

Appendix C. Quantization Process

Quantizing the network means converting it to use reduced bit-width representations for weights and/or activations. This saves model size and allows higher throughput operations to be used on hardware platforms. For example, INT8 quantization, when converting from floating-point to integer values, has a quantization process that multiplies the floating-point value by some scaling factor and then rounds the result to an integer. Different quantization methods differ in the way they determine the scaling factors. On the other hand, for weights, which are known during the inference, they are converted in advance and stored in the target format (e.g., INT8).

Fig. 10(a) depicts the dynamic quantization process. The scaling factor for activation is determined dynamically based on the range of data observed on the fly.

Fig. 10(b) depicts the static quantization process. Calibration is performed on a representative data set to determine the best quantization parameters for the activation. Compared to dynamic quantization, the running process of calculating the scaling factor (specifically, finding the maximum and minimum values and then performing the division operation) is avoided, and no dynamic adjustment of the parameters is required, saving a significant computational overhead.

Fig. 10(c) depicts the process of our proposed AFP quantization, where we simplify the quantization and MAC operation.Based on our AFP data format, the quantization is implemented by performing shift and logic operations, and the multiplication operation of MACs is implemented by adding exponents, which reduces the power during the inference process.

Appendix D. Compare with POSIT format

The posit format is proposed in an attempt to achieve the same representation as FP32. As shown in Fig. 11, the posit format consists of four parts: the sign bit, the regime bits, the exponent bits, and the fraction bits. The representa-

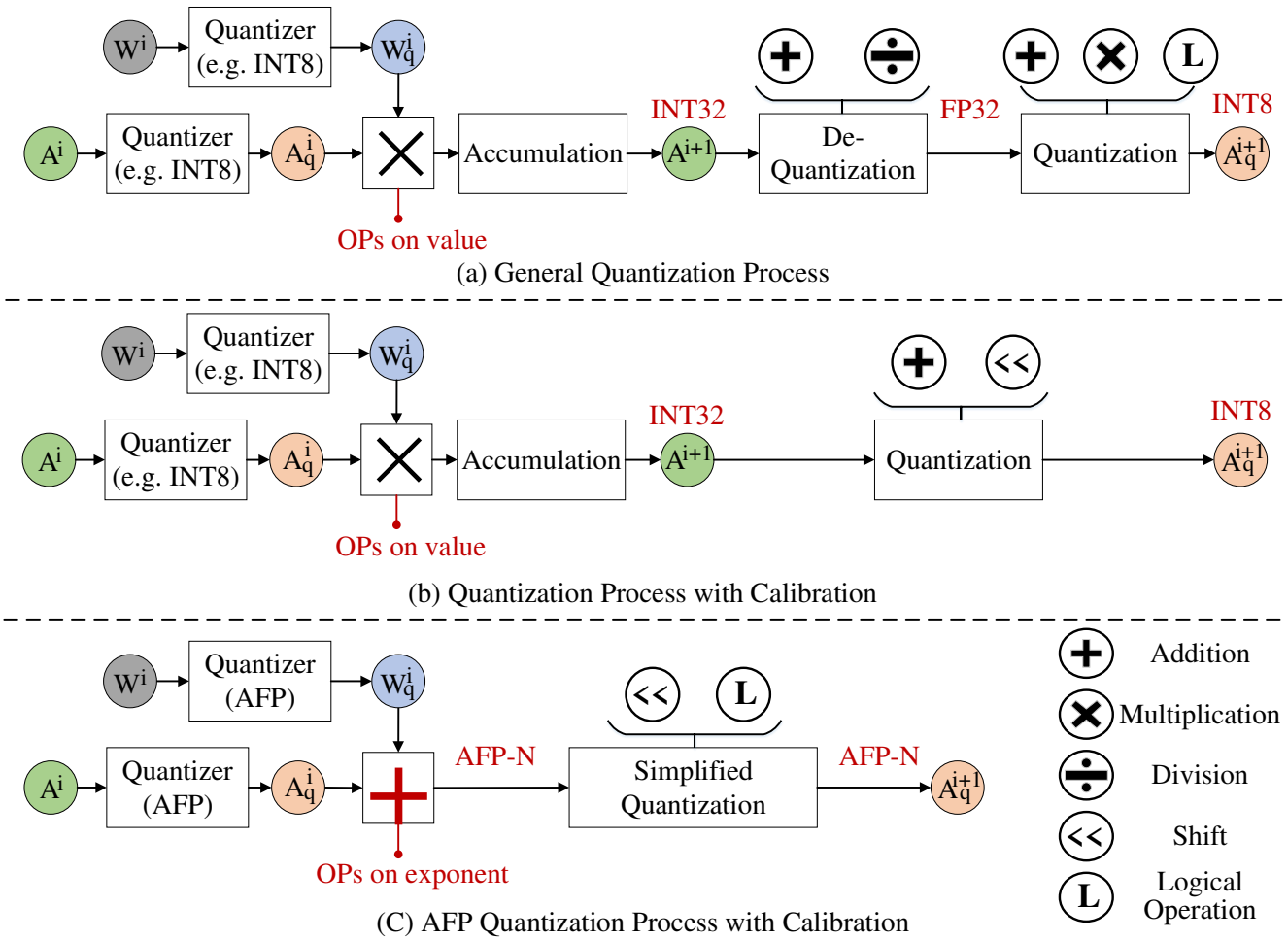


Figure 10: The process of the quantization.

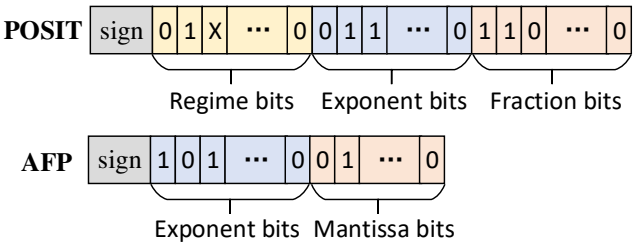


Figure 11: Comparison numeric format with POSIT and AFP.

tion range of posit is adjusted by the regime bits, but its bit-width is limited by the extra regime bits, which cannot be too small. Besides, each bit in the regime bits contains three possibilities: '0', '1' and 'x', where 'x' means don't care. This makes it unfriendly for existing hardware platforms because of the many extra logical judgments. Compared to POSIT, our scheme is a variant of conventional floating-point presentation, which includes only sign bits, exponent bits, and mantissa bits. Our AFP encode the DNN parameters in a more hardware-friendly and efficient manner to reduce the overhead of quantization and MAC operations during the inference.