Hardware-Efficient Quantization for Green Custom Foundation Models

Toshiaki Koike-Akino * [1] [2] **Chang Meng** [2] **Volkan Cevher** [2] **Giovanni De Micheli** [2]

Abstract

We propose a new hardware-efficient quantization (HEQ) for low-power full-custom foundation models. The HEQ jointly optimizes multiplier hardware and weight quantization to minimize the total power consumption. Exploiting power profile of custom multipliers, our method achieves a significant power reduction up to 20 folds.

1. Introduction

Recent foundation models exhibit excellent performance for a variety of artificial-intelligence (AI) domains such as language and vision processing. However, the required storage and computational resources are ever growing every year (Schwartz et al., 2020). To solve the issue, various model compression methods for green AI such as distillation, pruning, and quantization, were introduced (Frantar & Alistarh, 2022; Gou et al., 2021; Reed, 1993; Yuan & Agaian, 2023; Gholami et al., 2022).

For instance, binarized networks (Qin et al., 2023; Courbariaux et al., 2016; Kim & Smaragdis, 2016; Rastegari et al., 2016; Zhou et al., 2016; Liu et al., 2018) have shown relatively good performance even with 1-bit quantization. Nevertheless, binarization often causes a substantial loss, and most quantization papers (Frantar et al., 2022; Lin et al., 2023) for foundation models consider at least 3 bits to achieve an acceptable performance.

DeepShift (Elhoushi et al., 2021) uses power-of-two weights to eliminate multiplication operations. Hessian-aware quantization (HAWQ) (Dong et al., 2019) uses layer-wise quantization based on optimal brain pruning (LeCun et al., 1989). It was extended to mixed-precision and dyadic rationals (Dong et al., 2020; Yao et al., 2021). Then, GPTQ (Frantar et al., 2022) extends them with zero-shot calibration, while activation-aware quantization (AWQ) (Lin et al., 2023) uses adaptive scaling.

[1]Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139, USA [2]Ecole Polytechnique Federale de Lausanne (EPFL), 1015 Lausanne, Switzerland. Correspondence to: Toshiaki Koike-Akino <koike@merl.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Our paper provides a new framework for hardware-efficient quantization (HEQ), exploiting the synthesis profile of custom floating-point (FP) multipliers. The key idea is based on the fact that the power consumption of multipliers depends on weight distributions, motivating us to optimize the weight quantization to implement green AI chips, i.e., low-power custom foundation models. The major contributions of this paper are listed as follows.

- We synthesize the hardware of FP multipliers, to show its energy efficiency over integer multipliers.
- We propose an HEQ framework, enabling hardware profiles differentiable to optimize the weight quantization for power reduction.
- Our HEQ framework achieves 25% power reduction, and our custom multipliers provide up to 20-fold power reduction altogether.

2. Green Foundation Hardware Design

2.1. Floating-Point vs. Integer Quantization

Floating-point (FP) format is defined by 3 parameters $(N_\mathrm{e}, N_\mathrm{m}, B)$: the number of exponent bits, number of mantissa bits, and bias, respectively. Let s, e, and m be the 1-bit sign, N_e-bit exponent, and N_m-bit mantissa, respectively. The total bit width is thus $n = 1 + N_\mathrm{e} + N_\mathrm{m}$. We represent e and m as both integers and binary strings, interchangeably, without losing generality. The FP value is defined as: $x = (-1)^s \times 2^{E-B} \times M$, where $E = e$ and $M = 1.m$ when $e \neq 0$, unless otherwise $E = 1$ and $M = 0.m$ (a.k.a., subnormal condition). FP format is generic enough to cover integers, power-of-twos, dyadic numbers, and fixed-point formats. FP8 format was rigorously analyzed for deep learning (Noune et al., 2022), and pytorch supports `float8_e5m2` and `float8_e4m3`, for $(N_\mathrm{e}, N_\mathrm{m}, B) = (5, 2, 15)$ and $(4, 3, 7)$, respectively. Likewise, we denote $(N_\mathrm{e}, N_\mathrm{m}, B)$ FP format as "eN_emN_mbB". Unless specified, bias is chosen as $B = 2^{N_\mathrm{e}-1} - 1$.

Most standard n-bit integer quantization method (Gholami et al., 2022) uses group-wise scale and bias: $W_\mathrm{q} = W_\mathrm{int} S + Z$, where integer value is $W_\mathrm{int} = \mathrm{round}[(W - Z)/S]$, scale value is $S = (W_\mathrm{max} - W_\mathrm{min})/(2^n - 1)$ and bias is $Z = W_\mathrm{min}$ with W_max and W_min being the maximum and

Table 1. Power/delay/area profiles of general multipliers designed through Yosys/ABC logic synthesis and Synopsys Design Compiler on 45nm CMOS technology standard cell library. Power consumption is at 0.2GHz clock frequency.

Multipliers	int32	float32$_{e8m23}$	int16	float16$_{e5m10}$	bfloat16$_{e8m7}$	int8	float8$_{e5m2}$	float8$_{e4m3}$	int4	float4$_{e3m0b6}$
Power (μW)	5,883.5	**4,886.3**	1,054.6	814.6	**435.6**	170.5	**63.3**	101.3	15.6	**8.4**
Delay (ns)	4.99	5.00	2.67	3.76	3.25	1.58	1.25	1.65	0.45	0.29
Area (μm^2)	5,412.8	4,063.9	1,157.6	828.9	508.6	231.2	95.2	144.7	29.5	16.0

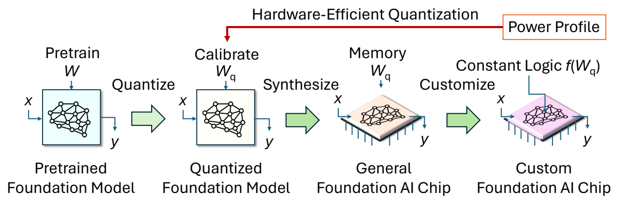

Hardware-Efficient Quantization
Power Profile
Pretrain
W
Quantize
Calibrate
W_q
Synthesize
Memory
W_q
Constant Logic $f(W_q)$
Customize
x
y
Pretrained Foundation Model
Quantized Foundation Model
General Foundation AI Chip
Custom Foundation AI Chip

Figure 1. Design of green custom foundation models.

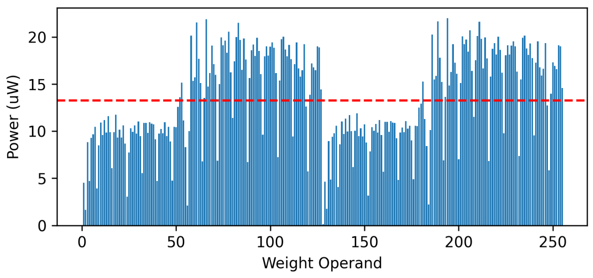


Figure 3. Power profile across quantized weight value for custom FP8 e4m3 multipliers. Average power is 13.3μW, average delay is 0.48ns, and average area is 28.7μm^2.

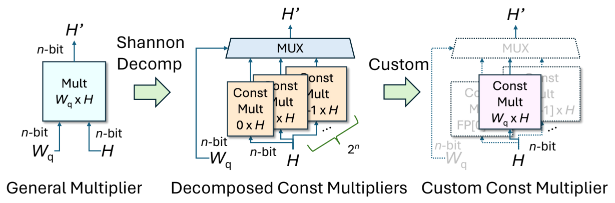


Figure 2. Shannon decomposition of general multiplier towards custom constant-weight multiplier.

minimum values of weights W in a group. Besides the n-bit integer $W_{\rm int}$, it requires extra bits for scale and bias, which are typically stored as FP values. For fair comparison, we do not consider extra bits for group-wise scale and bias.

2.2. Floating-Point vs. Integer Multipliers

As weight multiplications in AI models are the most hardware intensive part, we analyze the power, delay, and area profiles of multiplier hardware. We synthesize general multipliers (George & Tomar, 2019; Yadav et al., 2022) with the Nangate 45nm Open Cell Library (Nangate, 2011) using Yosys (Wolf et al., 2013), ABC (Brayton & Mishchenko, 2010), and Synopsys Design Compiler (Synopsys, 2024). More details are described in Appendix B, C, and D.

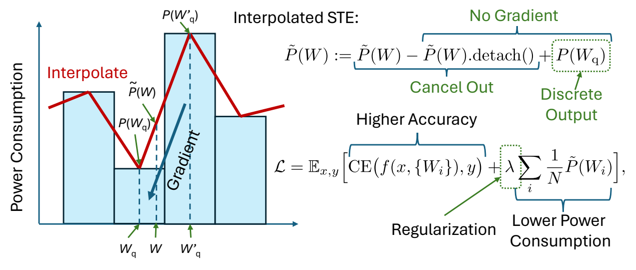
Table 1 shows the comparisons of different precision multipliers. Notably, it reveals that FP multipliers are hardware-efficient compared to integer multipliers. It is because the major block in FP multipliers is an integer multiplier block for mantissa parts, which requires quadratic complexity $\mathcal{O}[N_{\rm m}^2]$. For example, FP32 multipliers use a 24-bit integer multiplier, which is simpler than the full 32-bit integer multiplier. Accordingly, `bfloat16` having 7-bit mantissa is about **2-fold** energy-efficient over `float16` and `int16`. Similarly, `float8_e5m2` having just 2-bit mantissa is about **3-fold** energy-efficient than `int8` multipliers. Many integer quantization papers in the literature made a false assumption that FP operations are more complex than integer operations. Unlike such papers, we focus on FP quantization to realize green AI models. More analysis of general multiplier design for various precision is found in Appendix D.

2.3. Custom Green AI Deployment

Figure 1 illustrates a design framework of green custom foundation AI models. The foundation model is initially pretrained using training datasets. The model weight W is then quantized to $W_{\rm q}$. This quantization may employ a calibration step using calibration datasets to minimize the quantization loss, similar to (Frantar et al., 2022; Lin et al., 2023). Synthesizing the model provides a general AI chip, which uses general multipliers to compute a product of arbitrary hidden node H and quantized weight $W_{\rm q}$. This general AI chip reads a built-in memory storing the quantized weight $W_{\rm q}$. Our custom AI chip uses a constant logic and custom multipliers dedicated to each hidden node to be multiplied with a constant weight $W_{\rm q}$ without accessing memory. Once the weight is quantized and frozen, we no longer need general FP multipliers as one of operands is always constant. Our HEQ approach takes the power profile of such a custom AI model into account to calibrate the quantized weights.

2.4. Constant-Weight Multiplier

Any boolean functions can be factorized into fewer-bit boolean functions via Shannon decomposition (Shannon,



$$\tilde{P}(W) := \underbrace{\tilde{P}(W) - \tilde{P}(W).\text{detach}()}_{\text{Cancel Out}} + \boxed{P(W_q)}$$

$$\mathcal{L} = \mathbb{E}_{x,y}\Big[\text{CE}(f(x, \{W_i\}), y) + \lambda \sum_i \frac{1}{N} \tilde{P}(W_i)\Big],$$

Figure 4. Interpolated STE for differentiable hardware profile. Regularized loss to minimize cross entropy and power consumption.

1949). Figure 2 shows such a general multiplier decomposed into 2^n constant multipliers. It is obvious that each constant multiplier is much simpler than the original general multiplier because it has only one operand and another multiplicand is constant. For custom AI chips, we do not need the multiplexer to select the output from multiple constant multipliers because the quantized weight W_q is determined at the synthesis time.

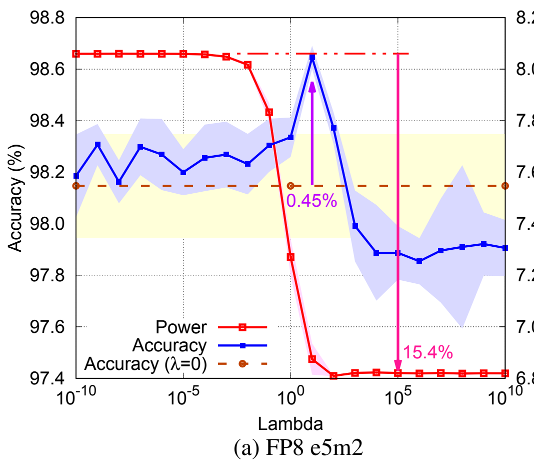
This custom multiplier can significantly reduce the power consumption. Figure 3 shows the power profile for the custom FP8 e4m3 multipliers over constant weight W_q across 2^8 quantization choices. Its average power consumption is 13.3μW, which is **7.6-times** lower power than the general FP8 e4m3 multiplier (101.3μW as in Table 1). More interestingly, it is even lower power than general `int4` multiplier (15.6μW). The power, delay and area profiles of other custom multipliers are found in Appendix F.

As shown in Figure 3, the power profile of custom multiplers highly depends on the quantization value W_q. For instance, if W_q is power-of-two having zeros in mantissa parts, the constant-weight multiplier will be a constant bit-shift operation. The power dip appears every 2^{N_m} quantization point in Figure 3, which is mainly because of this reason. This unequal power profile of custom multipliers motives us to optimize the quantized weight distribution.

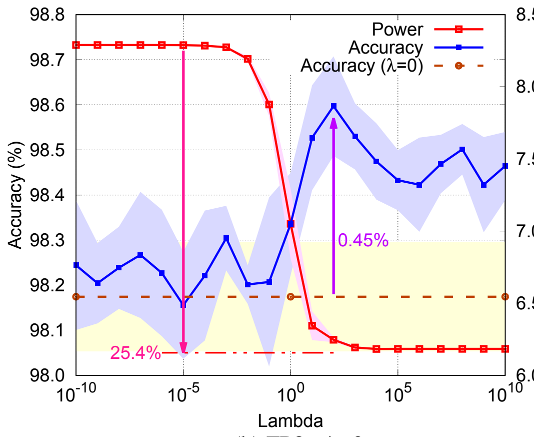
3. Hardware-Efficient Quantization (HEQ)

3.1. Quantization-Aware Training (QAT)

Our HEQ uses quantization-aware training (QAT) in calibration step, where the pretrained foundation model having full-precision weights W is quantized with a calibration dataset. QAT calibration refines the weights W such that its quantized value W_q maintains a good performance. Because quantization is non-differentiable, we use *straight-through estimator* (STE) trick (Gholami et al., 2022). Specifically, the forward processing is carried out as follows: $W := W - W.\text{detach}() + W_\text{q}$, where detach() means stopping the gradient in the backward pass, and quantized weight W_q is obtained by casting FP32 value to lower-precision FP.



(a) FP8 e5m2



(b) FP8 e4m3

Figure 5. Power-aware quantization results across regularization factor λ. Error band shows a confidence interval under one standard deviation over 7 random seeds.

Although this casting process is not differentiable, the STE trick can bypass the gradient calculation.

The weights are updated with gradient methods to minimize a loss function such as cross entropy in the calibration step. With the QAT calibration, quantization error can be significantly reduced compared to post-training quantization (PTQ), which directly casts the FP32 weights into the closest value in the lower-precision FP format. Nevertheless, the QAT alone does not efficiently promote the energy efficiency unless we can control the weight distribution.

3.2. Hardware-Aware Calibration

To further reduce the power consumption, our HEQ uses a regularized loss function to minimize the cross entropy loss and the total power consumption at once. We first obtain the synthesis profile of custom multipliers like in Figure 3, which provides a power consumption table $P(W_\text{q})$ depending on 2^n choices of quantization value W_q. As such a look-up table is not differentiable, we apply a linear

Table 2. Comparison of quantization methods for implementing custom ViT model.

	PTQ on General Multiplier									
Precision	$\text{FP32}_{\text{e8m23}}$	$\text{FP16}_{\text{e5m10}}$	$\text{BF16}_{\text{e8m7}}$	FP8_{e5m2}	FP8_{e4m3}	$\text{FP6}_{\text{e3m2b7}}$	$\text{FP5}_{\text{e3m1b7}}$	$\text{FP4}_{\text{e3m0b6}}$	$\text{INT4}_{\text{e0m3b4}}$	$\text{FP3}_{\text{e2m0b5}}$
Accuracy (%)	$98.29_{\pm 0.11}$	$98.03_{\pm 0.21}$	$98.04_{\pm 0.22}$	$98.01_{\pm 0.25}$	$97.81_{\pm 0.25}$	$97.83_{\pm 0.00}$	$97.45_{\pm 0.00}$	$92.49_{\pm 0.90}$	$10.69_{\pm 1.25}$	$14.25_{\pm 2.06}$
Power (μW)	4,886.3	814.6	435.6	63.3	101.3	46.62	24.04	8.4	15.6	1.2

	HEQ on Custom Multiplier									
Precision	$\text{FP32}_{\text{e8m23}}$	$\text{FP16}_{\text{e5m10}}$	$\text{BF16}_{\text{e8m7}}$	FP8_{e5m2}	FP8_{e4m3}	$\text{FP6}_{\text{e3m2b7}}$	$\text{FP5}_{\text{e3m1b7}}$	$\text{FP4}_{\text{e3m0b6}}$	$\text{INT4}_{\text{e0m3b4}}$	$\text{FP3}_{\text{e2m0b5}}$
Accuracy (%)	—	$98.70_{\pm 0.09}$	—	$98.65_{\pm 0.05}$	$98.60_{\pm 0.11}$	$\mathbf{98.78}_{\pm 0.05}$	$98.67_{\pm 0.09}$	$97.99_{\pm 0.08}$	$55.91_{\pm 6.74}$	$97.35_{\pm 0.14}$
Power (μW)	—	$179.09_{\pm 0.82}$	—	$6.87_{\pm 0.06}$	$6.25_{\pm 0.02}$	$2.35_{\pm 0.00}$	$1.19_{\pm 0.01}$	$0.60_{\pm 0.00}$	$0.13_{\pm 0.00}$	$\mathbf{0.07}_{\pm 0.00}$

interpolation for arbitrary precision W to get interpolated power consumption:

$$\tilde{P}(W) = P(W_\text{q})\frac{W'_\text{q}-W}{W'_\text{q}-W_\text{q}} + P(W'_\text{q})\frac{W-W_\text{q}}{W'_\text{q}-W_\text{q}}, \quad (1)$$

where W_q and W'_q are the closest and second closest quantization values from W. Then, we further introduce the STE trick on top of this interpolated profile as follows:

$$\tilde{P}(W) := \tilde{P}(W) - \tilde{P}(W).\text{detach}() + P(W_\text{q}), \quad (2)$$

which we call interpolated STE trick, enabling discrete output of power consumption yet differentiable. This interpolated STE trick for differentiable hardware profile is depicted in Figure 4. The gradient towards lower-power quantization value can promote reducing the total power consumption. Different interpolation could be used as well.

In order to balance the trade-off between the power reduction and accuracy, our HEQ uses a regularized loss function:

$$\mathcal{L} = \mathbb{E}_{x,y}\big[\text{CE}\big(f(x,\{W_i\}),y\big) + \lambda\textstyle\sum_i \frac{1}{N}\tilde{P}(W_i)\big], \quad (3)$$

where $\text{CE}(\cdot)$ denotes the cross entropy, (x, y) is a pair of supervised calibration data input and output, $f(\cdot)$ denotes the foundation model, $\{W_i\}$ is a set of all N weights, W_i is the i-th weight, and λ is a regularization factor.

4. Experiments

We customize the vision transformer (ViT) model (Dosovitskiy et al., 2020) pre-trained on ImageNet-21k (Deng et al., 2009), and further fine-tuned towards ImageNet-1k and CIFAR10 dataset (Krizhevsky et al., 2009). This ViT model has a total of 86.6M parameters to quantize. More details of settings are described in Appendix E.

Figure 5 shows classification accuracy and power consumption across regularization factor λ for custom FP8 multipliers e5m2 and e4m3. Interestingly, a moderate regularization offers improvement in both classification accuracy and energy efficiency by up to 0.45% and 25.4%, respectively. Appendix G and I show more results on fewer-bit FP quantizations, showing importance of bias factor and exponent bit allocation. How the weight distribution is adjusted by regularization is also discussed in Appendix H.

Table 2 shows the overall quantization results. The full-precision FP32 has 98.29% accuracy, and PTQ with lower precision gradually degrades the performance. FP8 still maintains a reasonable accuracy above 97.81%, while FP4 has about 6% loss. Nevertheless, FP4 is still much better than INT4 quantization, which has a poor performance of 10.69%. More importantly, general FP4 multiplier has 582-fold power reduction over FP32.

Our HEQ with proper λ can improve the performance and energy efficiency. Notably, FP5 quantization has even better performance than FP32 precision due to regularization gain at only 1.19μW power consumption, which is **4100-times** lower energy $(4886.3/1.19)$. The reason why FP5 and FP6 are better than higher-precision FP is discussed in Appendix G and H. Using the custom multipliers, the power consumption for each FP quantization can be reduced by up to **20 folds** (e.g., $24.04\mu\text{W}/1.19\mu\text{W}$ for FP5). When using FP3 quantization, the total power consumption over the entire 86.6M-parameter model is still only 6.06W, while keeping the quantization loss within 1% from FP32, which requires an enormous power of 0.42MW with fully parallelized general multipliers. Thus, our full-custom AI chip can be greater than **4 orders of magnitude** greener $(0.42\text{M}/6.06 \simeq 7 \times 10^4)$. The power-accuracy tradeoff is highlighted in Appendix J. We also evaluated delay-aware quantization using our HEQ framework in Appendix K, achieving up to **19-times** acceleration in inference time.

5. Conclusion

We proposed hardware-efficient quantization (HEQ), achieving a significant power reduction. With Shannon decomposition, our custom multipliers exhibit a few orders of magnitude lower-power consumption than general multipliers. We introduced an interpolated STE trick to make hardware profiles differentiable, leading to a significant improvement in both accuracy and energy efficiency. It allows 3-bit quantization having 4 orders of magnitude power reduction over FP32 models while the loss is within 1%. We plan to extend to mixed-precision and approximated logic synthesis with more rigorous experiments of various foundation models.

Impact Statement

Our paper's goal is to advance the field of machine learning. While there are many potential societal consequences of our work, we feel none of major negative impacts is specifically highlighted.

References

Brayton, R. and Mishchenko, A. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, pp. 24–40. Springer, 2010.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dong, Z., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 293–302, 2019.

Dong, Z., Yao, Z., Arfeen, D., Gholami, A., Mahoney, M. W., and Keutzer, K. HAWQ-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529, 2020.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Elhoushi, M., Chen, Z., Shafiq, F., Tian, Y. H., and Li, J. Y. DeepShift: Towards multiplication-less neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2359–2368, 2021.

Frantar, E. and Alistarh, D. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488, 2022.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

George, M. L. and Tomar, G. S. Comparative review of floating point multiplier. *International Journal of Hybrid Technology*, 12(2):21–48, 2019.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pp. 291–326. Chapman and Hall/CRC, 2022.

Gou, J., Yu, B., Maybank, S. J., and Tao, D. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

Kim, M. and Smaragdis, P. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images, 2009.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. AWQ: Activation-aware weight quantization for LLM compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Lin, M., Ji, R., Xu, Z., Zhang, B., Wang, Y., Wu, Y., Huang, F., and Lin, C.-W. Rotated binary neural network. *Advances in neural information processing systems*, 33: 7474–7485, 2020.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. *Advances in neural information processing systems*, 30, 2017.

Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., and Cheng, K.-T. Bi-Real Net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 722–737, 2018.

Nangate. The Nangate 45nm Open Cell Library. `https://si2.org/open-cell-library/`, 2011.

Noune, B., Jones, P., Justus, D., Masters, D., and Luschi, C. 8-bit numerical formats for deep neural networks. *arXiv preprint arXiv:2206.02915*, 2022.

Qin, H., Gong, R., Liu, X., Shen, M., Wei, Z., Yu, F., and Song, J. Forward and backward information retention for accurate binary neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2250–2259, 2020.

Qin, H., Zhang, M., Ding, Y., Li, A., Cai, Z., Liu, Z., Yu, F., and Liu, X. BiBench: Benchmarking and analyzing network binarization. In *International Conference on Machine Learning*, pp. 28351–28388. PMLR, 2023.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016.

Reed, R. Pruning algorithms—a survey. *IEEE transactions on Neural Networks*, 4(5):740–747, 1993.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

Shannon, C. E. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949.

Synopsys. Design Compiler. `https://www.synopsys.com/`, 2024.

Wolf, C., Glaser, J., and Kepler, J. Yosys - A free Verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, 2013.

Yadav, J., Kumar, A., Shareef, S., Bansal, S., and Rathour, N. Comparative analysis of Vedic multiplier using various adder architectures. In *Journal of Physics: Conference Series*, volume 2327, pp. 012022. IOP Publishing, 2022.

Yao, Z., Dong, Z., Zheng, Z., Gholami, A., Yu, J., Tan, E., Wang, L., Huang, Q., Wang, Y., Mahoney, M., et al. HAWQ-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pp. 11875–11886. PMLR, 2021.

Yuan, C. and Agaian, S. S. A comprehensive review of binary neural network. *Artificial Intelligence Review*, 56 (11):12949–13013, 2023.

Zhang, Y., Pan, J., Liu, X., Chen, H., Chen, D., and Zhang, Z. FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 171–182, 2021.

Zhao, R., Song, W., Zhang, W., Xing, T., Lin, J.-H., Srivastava, M., Gupta, R., and Zhang, Z. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 15–24, 2017.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

A. Related Work

Various model compression methods were proposed, such as knowledge distillation (Frantar & Alistarh, 2022; Gou et al., 2021), pruning (Reed, 1993), and quantization (Yuan & Agaian, 2023; Gholami et al., 2022).

Extreme 1-bit quantization (Qin et al., 2023) has shown relatively good performance even with 1-bit quantization, e.g., BNN (Courbariaux et al., 2016; Kim & Smaragdis, 2016), XNOR-Net (Rastegari et al., 2016), DoReFa-Net (Zhou et al., 2016), Bi-Real (Liu et al., 2018), IR-Net (Qin et al., 2020), RBNN (Lin et al., 2020), and ABC-Net (Lin et al., 2017). They proposed a variety of gradient approximation methods for non-differentiable binarization operation. For example, BNN uses STE trick as:

$$W := \underbrace{\mathrm{sign}(W)}_{\text{Non-Differentiable}} + \underbrace{\overbrace{W'}^{\text{Differentiable}} - W'.\mathrm{detach}()}_{\text{Cancel Out}}, \tag{4}$$

$$W' = \mathrm{clamp}(W, -1, 1), \tag{5}$$

where $\mathrm{sign}(x) = (-1)^{x \le 0}$ is non-differentiable sign function, and $\mathrm{clamp}(x, a, b) = \max(a, \min(b, x))$ is a piece-wise differentiable truncation function.

Binarized networks have many papers on hardware implementation, mostly on generic field-programmable gate-array (FPGA) platform, e.g., (Zhao et al., 2017; Zhang et al., 2021). Nevertheless, binarization often causes a substantial loss, and most quantization papers (Frantar et al., 2022; Lin et al., 2023) for foundation models consider at least 3 bits to achieve an acceptable performance.

DeepShift (Elhoushi et al., 2021) uses power-of-two weights to eliminate multiplication operations. Hessian-aware quantization (HAWQ) (Dong et al., 2019) uses layer-wise quantization based on optimal brain pruning (LeCun et al., 1989). HAWQv2 (Dong et al., 2020) considers mixed-precision weight and activation. HAWQv3 (Yao et al., 2021) uses integer weight in dyadic format. Then, GPTQ (Frantar et al., 2022) extends HAWQ using zero-shot calibration, while activation-aware quantization (AWQ) (Lin et al., 2023) uses activation-dependent scaling.

B. Standard Cell Library

In integrated circuit design, a standard cell library is a collection of pre-designed and pre-characterized logic gates and other standard circuit elements used to implement a circuit, ensuring consistency and efficiency in the design and manufacturing process. Specifically, we utilize the Nangate 45nm open cell library (Nangate, 2011), January 2011 version, in our experiments. The library contains 134 standard cells including:

- 9 non-functional cells;
- 16 flip-flops;
- 5 latches;
- 102 combinational cells;
- half-adders and full-adders.

The cells offer basic logic functions (AND, NAND, OR, NOR, XOR, XNOR, OR-AND, AND-OR, BUFF, and INV) with different drive strengths.

We use a typical voltage of 1.1V at 25°C room temperature. Note that our analysis uses 45nm CMOS (complementary metal-oxide semiconductor) technology, while much lower power can be achieved with recent CMOS technologies such as the 5nm process.

C. Logic Synthesis

Logic synthesis converts an abstract specification of desired circuit behavior, typically at the register-transfer level (RTL), into a gate netlist consisting of cells from a standard cell library. This process optimizes for reducing circuit power, delay, and area. In our experiments, we generate custom multipliers using three logic synthesis tools, *i.e.*, Yosys (Wolf et al., 2013), ABC (Brayton & Mishchenko, 2010), and Synopsys Design Compiler (Synopsys, 2024) following the workflow described below.

First, we write RTL codes to describe the custom multiplier using Verilog. Yosys is then used to convert the Verilog code to a gate netlist via the `synth` command. Next, the obtained netlist is further refined by ABC. Specifically, we apply the area-oriented optimization script `compress2rs`, and the area-oriented technology mapping command `amap` to map the circuit into cells in the Nangate 45nm library.[1] To reduce the power, delay, and area more aggressively, the optimized netlist is further synthesized by Synopsys Design Compiler. We used `compile_ultra` command to further optimize the logic under a maximum delay constraint of 5ns. The power, delay, and area profiles are then analyzed, with a clock frequency of 0.2GHz to be used for the power analysis.

D. Hardware Profile of General Multiplier

We synthesize general multipliers (George & Tomar, 2019; Yadav et al., 2022) with the Nangate 45nm Open Cell Library (Nangate, 2011) using Yosys (Wolf et al., 2013), ABC (Brayton & Mishchenko, 2010), and Synopsys Design Compiler (Synopsys, 2024).

Figure 6 shows a general FP multiplier. It consists of 3 major blocks: mantissa multiplication, exponent addition,

[1]We apply area-oriented logic synthesis because area and power are postively correlated.

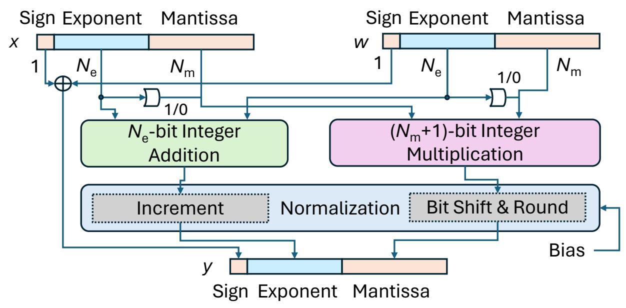


Figure 6. General FP multiplier diagram: exponent adder; mantissa multiplier; normalization. Hardware complexity is dominated by $(N_\mathrm{m} + 1)$-bit integer multiplier block.

and normalization. The most hardware-intensive block is mantissa multiplication in the order of $\mathcal{O}[N_\mathrm{m}^2]$ complexity. As the FP multiplier uses integer multipliers at a fewer-bit width, it can be more energy-efficient than full-bit-width integer multipliers. Other blocks in FP multipliers are in the linear complexity as they are based on integer additions and bit shifts.

Figure 7 shows the synthesis results in the power/delay/area profiles for few-precision multipliers over FP1 to FP8 having different N_e. As discussed, integer multipliers are more energy demanding, while power-of-two multipliers (exponential quantization with $N_\mathrm{m} = 0$) are most energy-efficient as no multiplication block is required in Figure 6.

E. ViT CIFAR10 Task

Table 3 lists hyperparameters for the experiment of ViT quantization. The base ViT model (`google/vit-base-patch16-224`)[2] is pretrained on ImageNet-21k and fine-tuned on ImageNet-1k. The ViT model is depicted in Figure 8. It has 12 layers of multi-head attention modules, each of which has 12 heads, 768 features, and a token length of 769. The original base model has 36.6M parameters in floating-point 32 bits, requiring 330MiB storage.

CIFAR10 is an image classification dataset having 10 classes of 32 × 32 colored images with 50k training samples and 10k test samples. We use up-sampling to 224 × 224 resolutions with random resized cropping and horizontal flip. The original classifier head has 1000 class output, and we selected 10 outputs based on the prediction score of CIFAR10 training data. The matched labels are listed in Table 4.

We use CIFAR10 training data for quantization calibration, and use testing data for performance analysis. AdamW is used with OneCycle scheduling up to 100 epochs while early stopping is taken place when accuracy dropped with a patience of 1.

[2]`https://huggingface.co/google/vit-base-patch16-224`

Table 3. Hyperparameter configurations for ViT quantization.

Hyperparameter	Value
Optimizer	AdamW
Learning Rate Schedule	OneCycle
Weight Decay	0.01
Batch Size	32
Epochs	100
Early Stopping Patience	1
Learning Rate	0.001

Table 4. Label matching from ImageNet-1k to CIFAR10.

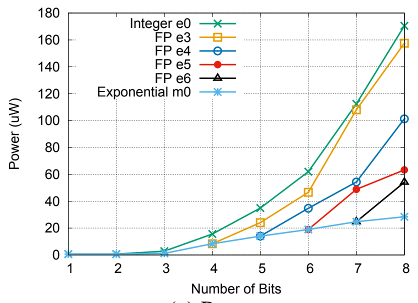
ImageNet-1k		CIFAR10	
404	airliner	0	airplane
675	moving van	1	automobile
9	ostrich	2	bird
285	Egyptian cat	3	cat
351	hartebeest	4	deer
152	Japanese spaniel	5	dog
32	tailed frog	6	frog
339	sorrel horse	7	horse
814	speedboat	8	ship
867	trailer truck	9	truck

We use our HEQ framework for 7 times with different random seeds to obtain the mean accuracy and mean power consumption as well as those standard deviations.

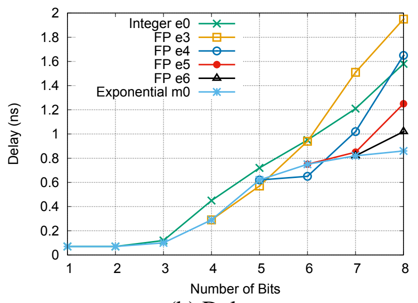
F. Hardware Profile of Custom Multiplier

Table 5 lists average power, delay, and area profiles for custom FP multipliers, i.e., constant-weight multipliers in the Shannon decomposition. When using general multipliers in Table 1, it is nearly infeasible to fully parallerize the foundation model. For instance, the ViT model having 86.6M parameters requires $4886.3\mu\mathrm{W} \times 86.6\mathrm{M} = 0.42\mathrm{MW}$ power and $4063.9\mu\mathrm{m}^2 \times 86.6\mathrm{M} = 0.35\mathrm{m}^2$ die size with FP32 general multipliers. However, the custom FP3 multipliers can pack the ViT in a dia size of about $0.53\mu\mathrm{m}^2 \times 86.8\mathrm{M} = 45\mathrm{mm}^2$, having $0.13\mu\mathrm{W} \times 86.6\mathrm{M} = 11.3\mathrm{W}$ power. Hence, realizing low-power full-custom foundation chip is feasible.

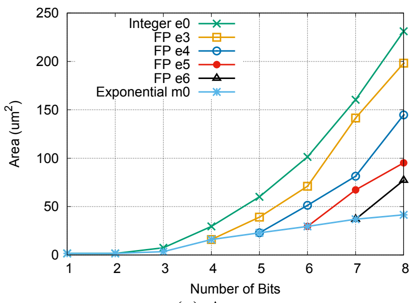
Figure 9 shows power, delay and area profiles of custom multipliers for FP8 e5m2. The power consumption highly depends on the quantized value of the weight operand for multiplications. When the weight operand has zeroes in mantissa part, the FP multiplier reduces to bit-shift operators and more energy-efficient than other weight values. Hence, one can see that the power dip happens every 4 apart when mantissa becomes 0.



(a) Power



(b) Delay



(c) Area

Figure 7. Power/delay/area profile of general multipliers on 45nm CMOS technology standard cell library.

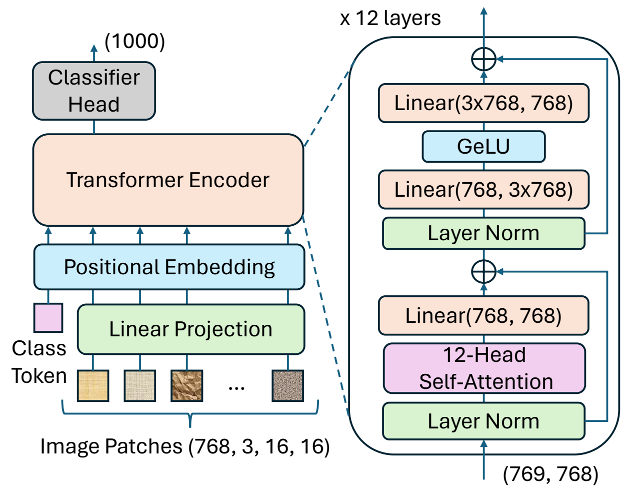


Figure 8. Base ViT foundation model.

Average power is 9.2μW, which is 7-fold lower than the power consumption of general FP8 multipliers requiring 63.3μW.

Figure 10 shows power, delay and area profiles of custom multipliers for FP8 e4m3. Similarly, the power dip happens every 8 apart when the mantissa bits become all 0. Average power is 12.9μW, which is 8-fold lower than the power consumption of general FP8 multiplier requiring 101.3μW.

Figure 11, 12, 13, 14 and 15 show hardware profiles for FP7 e4m2b8, FP6 e3m2b7, FP5 e3m1b7, FP4 e3m0b6, and FP4 e0m3b4 multipliers, respectively.

G. Linear vs. Nonlinear Quantization

FP quantization is generally nonlinear or non-uniform as the quantization step is non-equal. Nevertheless, FP quantization will be linear or uniform when we assign all bits into mantissa parts, i.e., $N_\mathrm{e} = 0$. When we assign all bits into exponent parts, it will be power-of-two quantization like DeepShift, i.e., $N_\mathrm{m} = 0$. When the precision is high

Table 5. Average power/delay/area profile of custom FP multipliers on 45nm CMOS technology standard cell library.

FP	Power (μW)	Delay (ns)	Area (μm^2)
FP16 e5m10	280.56	2.64	330.54
FP8 e5m2	9.23	0.41	21.30
FP8 e4m3	13.26	0.48	28.72
FP7 e4m2b8	7.53	0.34	17.24
FP6 e3m2b7	3.93	0.17	9.73
FP5 e3m1b7	2.32	0.10	6.10
FP4 e3m0b6	0.94	0.059	3.03
FP3 e2m0b5	0.13	0.015	0.53

like FP8, balancing bit allocations to exponent and mantissa parts can improve the overall performance. However, if there are only a few bits, how should we allocate them into exponent and mantissa?

Figure 16 shows QAT calibration results for FP3 to FP5 quantization using different exponent, mantissa, and bias parameters. It clearly shows that allocating all bits into the exponent part is more advantageous than the mantissa part for few-bit FP. This is favorable to hardware design as the power-of-two multipliers are simple bit-shift operators. It suggests that nonlinear/non-uniform quantization is generally better than linear/uniform quantization. For FP6, there is a slight improvement from e5m0 to e4m1 at the peak performance.

We can also find that the selection of bias parameter B is a crucial factor. Typically, FP bias is choise as $B = 2^{N_\mathrm{e}-1} - 1$ to balance the representation from large to small values. However, most AI weights have no large values, and we should increase the bias point in general. For example, the best bias for FP4 e3m0 was $B = 6$, which is larger than standard choice of $B = 3$.

H. Weight Distribution

The weight distributions are adjusted by the regularization factor λ. Figure 17 shows the weight distribution of custom

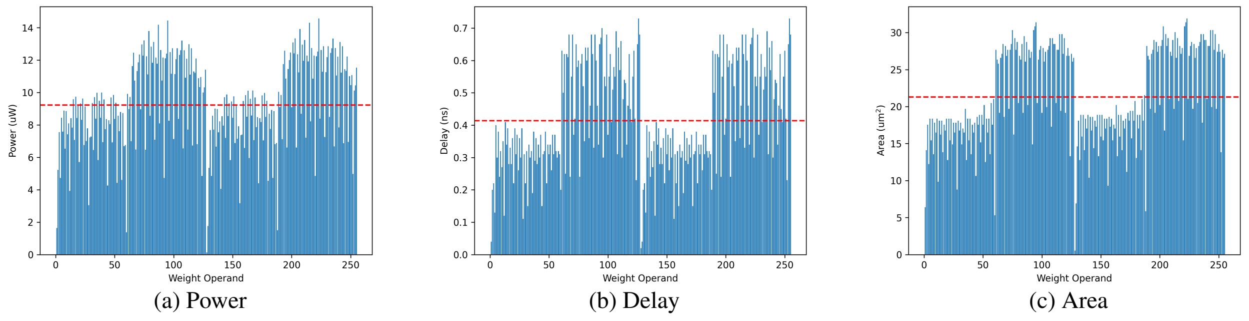


Figure 9. Power/delay/area profile across quantized weight value for custom FP8 multiplier: e5m2b15. Average power is 9.2μW, which is 7-fold lower than the power consumption of general FP8 multiplier requiring 63.3μW.

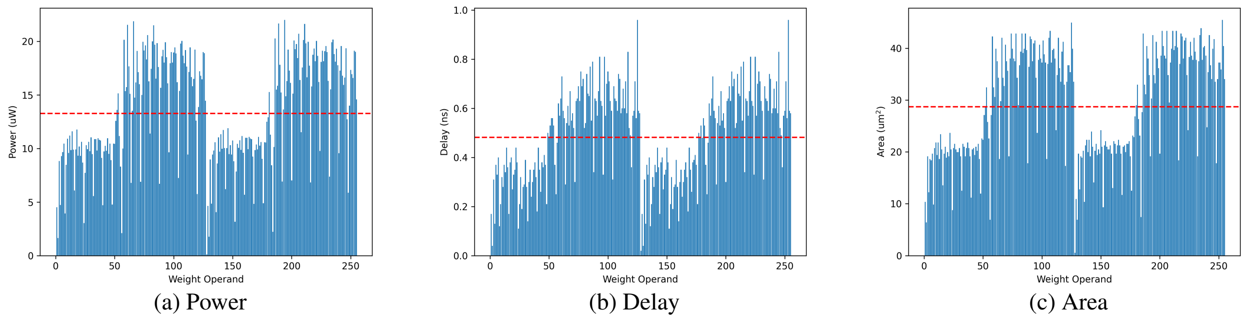


Figure 10. Power/delay/area profile across quantized weight value for custom FP8 multiplier: e4m3b7. Average power is 13.3μW, which is about 8-fold lower than the power consumption of general FP8 multiplier requiring 101.3μW.

multipliers for FP8 e4m3 when sweeping the regularization factor from $\lambda = 10^{-1}, 10^0, 10^1, 10^2, 10^3$. We can see that higher regularization provides more sparse weight distribution, where non-power-of-twos weights are gradually decreased. Balancing power-of-two weights and non-power-of-two weights at around $\lambda = 10^2$ can achieve higher accuracy and lower energy as discussed in power-aware quantization as shown in Figure 5. The reason why the moderate regularization improves accuracy is because of the weight regularization benefit, which can often improve the parameter efficiency for over-parameterized foundation models.

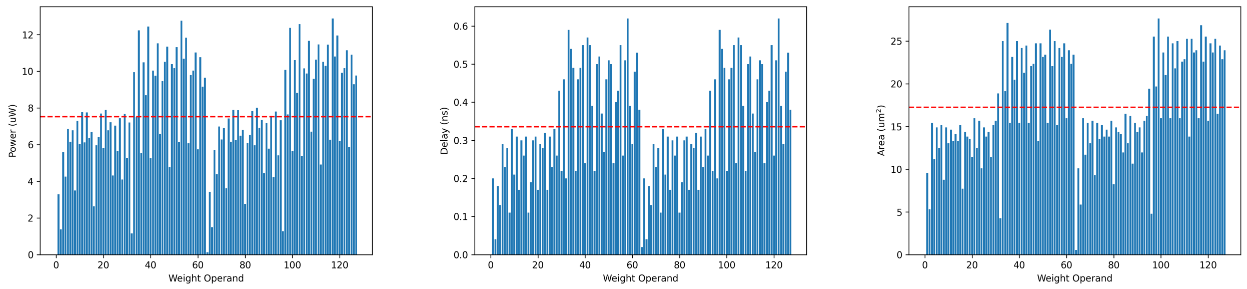
Figure 18 shows the weight distribution of custom multipliers for FP8 e5m2. We can see the similar trend: Larger the regularization factor λ, sparser the weight distributions. It is because custom multiplers having higher-power consumption are avoided. Nevertheless, too strong power reduction can degrade the performance, and the balanced regularization at around $\lambda = 10^1$ offers improved performance and energy efficiency jointly.

Note that FP8 quantization using a default bias $B = 2^{N_\mathrm{e}-1} - 1$ is not the best choice as unused quantization values exist as shown in those histogram figures. This is a major reason why FP6 and FP5 quantization were better than FP8 quantization. For fewer-bit FP quantizatins, we optimized the bias as shown in Appendix G. Accordingly, quantization values were well distributed as discssed below.

Figure 19, 20, 21, 22, and 23 show the weight histogram across the regularization factor λ for FP7 e4m2b8, FP6 e3m2b7, FP5 e3m1b7, FP4 e3m0b6 and FP4 e0m3b4 quantizations, respectively. As shown, the weight quantization is well distributed at low λ, and gradually sparsified with increased regularzation factor λ. FP4 e0m3b4 is equivalent to fixed-point precision or scaled `int4`. We observe that sparser distribution for FP4 e3m0, but not for INT4 case. The reason why INT4 has different behavior may be because the cross entropy term is very high and performance is poor like 55.91%.

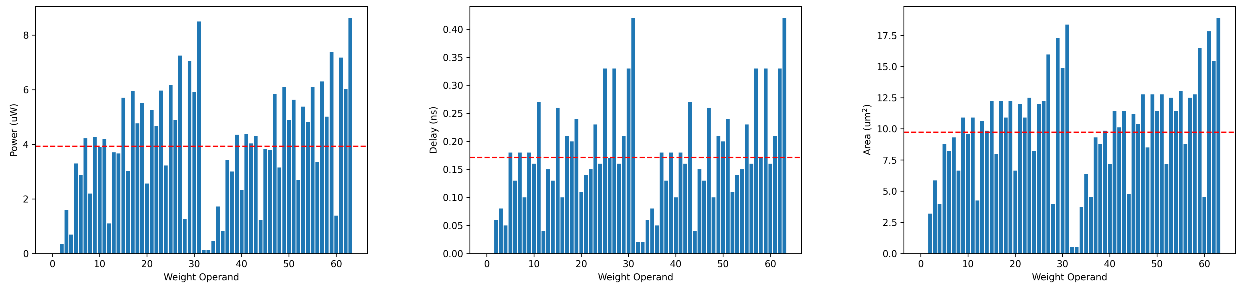
I. Power-Aware Quantization

The power-aware quantization in our HEQ framework for FP4 and FP3 is shown in this section. Figure 24 shows the accuracy and power profile across the regularization factor λ for FP4 e3m0b6, FP4 e0m3b5, and FP3 e2m0b5. As shown, higher regularization provides more energy efficient weight quantization. The accuracy is not necessary improved for such low-precision regimes, while we can keep a compara-



(a) Power (b) Delay (c) Area

Figure 11. Power/delay/area profile across quantized weight value for custom FP7 multiplier: e4m2b8.



(a) Power (b) Delay (c) Area

Figure 12. Power/delay/area profile across quantized weight value for custom FP6 multiplier: e3m2b7.

ble accuracy to the case of QAT with $\lambda = 0$.

Figure 25 shows the performance results of PTQ, QAT ($\lambda = 0$), and HEQ quantization methods across 1 to 8 bits. It shows PTQ has a large loss below 5-bit quantization, while QAT/HEQ maintains high accuracy above 2-bit quantization.

J. Accuracy-Power Tradeoff

When synthesizing the entire foundation model with 86.6M parameters, the total power will be enormous in general. Fig. 26 shows the classification accuracy vs. total power consumption tradeoff when implemented in fully-parallel AI chip. It presents performance of PTQ (on general multipliers), QTA ($\lambda = 0$ on general multipliers), and HEQ (with proper $lambda$ on custom multipliers). Our custom AI chip via HEQ framework has a feasible level of power consumption, keeping high accuracy. Whereas, the general AI chip using PTQ quantization requires infeasible level of high power.

Note that we focused on multiplier complexity, ignoring most other circuit complexity and the real tradeoff can be shifted, but we believe that our analysis provides an important step towards the real "green AI" achievement.

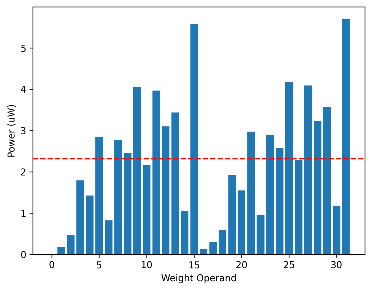
K. Delay-Aware Quantization

HEQ framework can also optimize the delay and area using the profile of custom multipliers as shown in Figure 9 and Figure 10. Figure 27 shows the delay-aware quantization results for FP8 e5m2 and e4m3 precision. Similar to power-aware quantization in Figure 5, we can achieve a significant improvement of processing delay as well as accuracy. For example, at a moderate regularization factor of $\lambda = 10^2$, the classification accuracy is improved by 0.5% and 0.35% for FP8 e5m2 and e4m3, respectively. More notably, the delay is significantly decreased by 19.5% and 34.5% by our HEQ framework. When comparing to a general AI chip with the general FP8 e3m4 multiplier (requiring 3.25ns), our custom AI chip (0.17ns) can achieve up to 19-fold speedup.

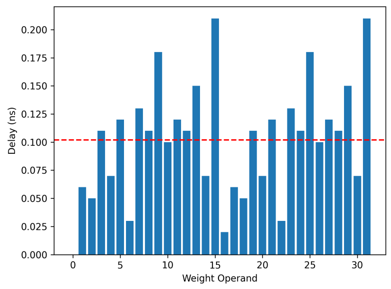
We can optionally include other hardware profiles in loss function jointly. For example, we may optimize the quantization distribution to minimize classification error, power consumption, delay, and area jointly by using more regularization factors:

$$\mathcal{L} = \mathbb{E}_{x,y}\Big[\mathrm{CE}\big(f(x, \{W_i\}), y\big) + \sum_i \frac{1}{N}\big(\lambda_1 \tilde{P}(W_i) + \lambda_2 \tilde{D}(W_i) + \lambda_3 \tilde{A}(W_i)\big)\Big], \quad (6)$$

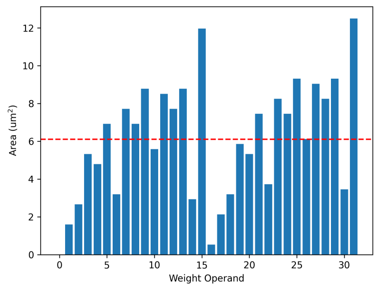
where λ_i are regularization factors, $\tilde{D}(\cdot)$ is an interpolated



(a) Power

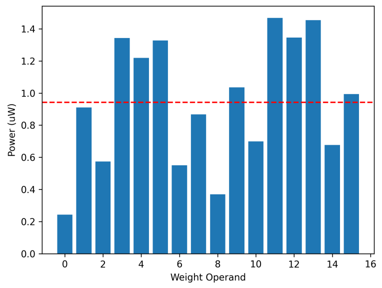


(b) Delay

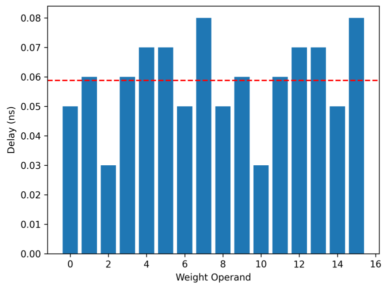


(c) Area

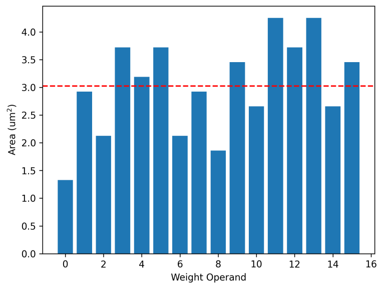
Figure 13. Power/delay/area profile across quantized weight value for custom FP5 multiplier: e3m1b7.



(a) Power



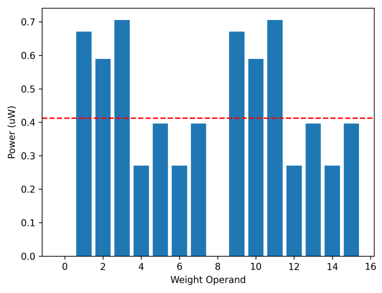
(b) Delay



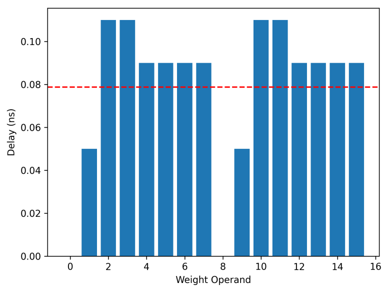
(c) Area

Figure 14. Power/delay/area profile across quantized weight value for custom FP4 multiplier: e3m0b6.

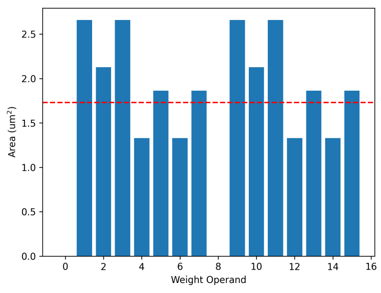
STE delay profile, and $\tilde{A}(\cdot)$ is an interpolated STE area profile.



(a) Power

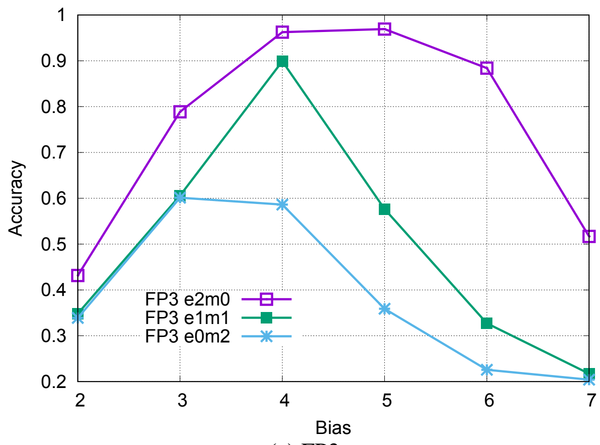


(b) Delay

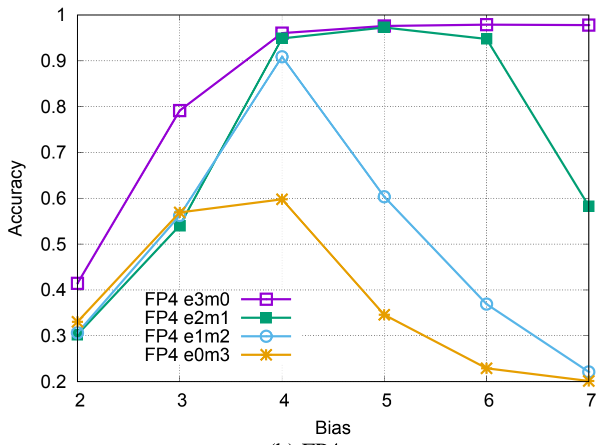


(c) Area

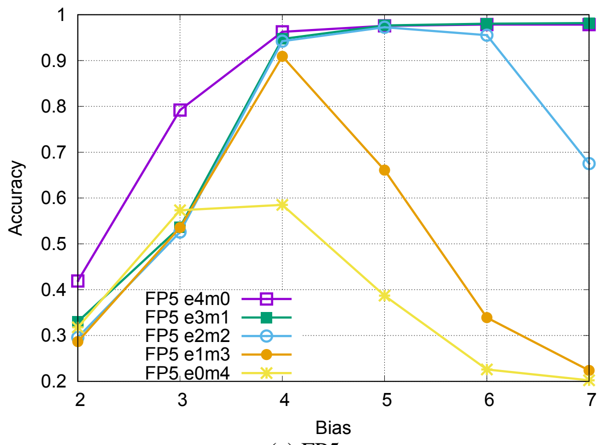
Figure 15. Power/delay/area profile across quantized weight value for custom FP4 multiplier: e0m3b4.



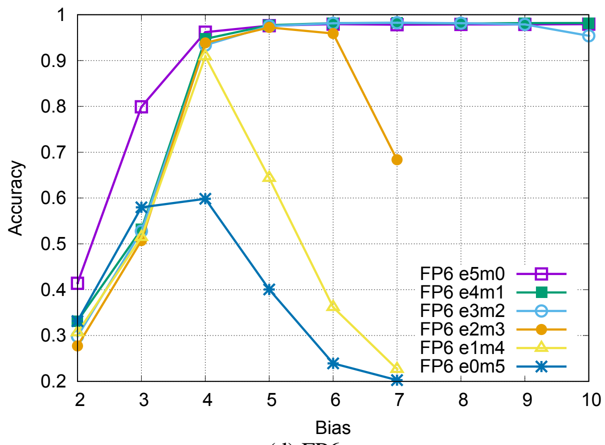
(a) FP3



(b) FP4

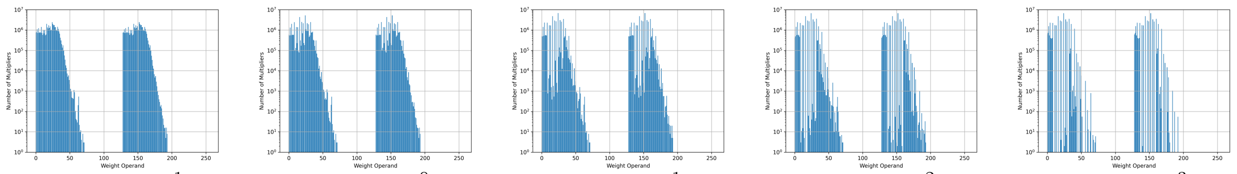


(c) FP5



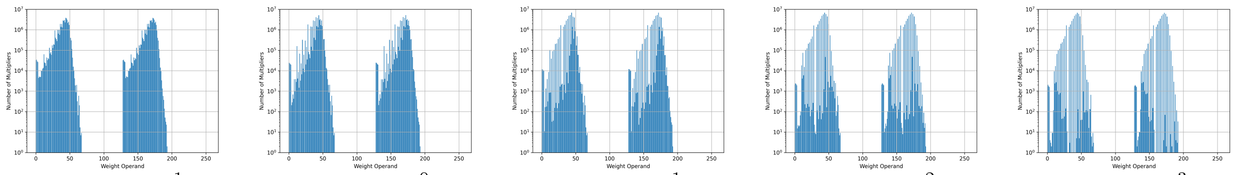
(d) FP6

Figure 16. QAT calibration performance for few-precision FP quantization, over different bit parameters ($N_{\rm e}$, $N_{\rm m}$, B).



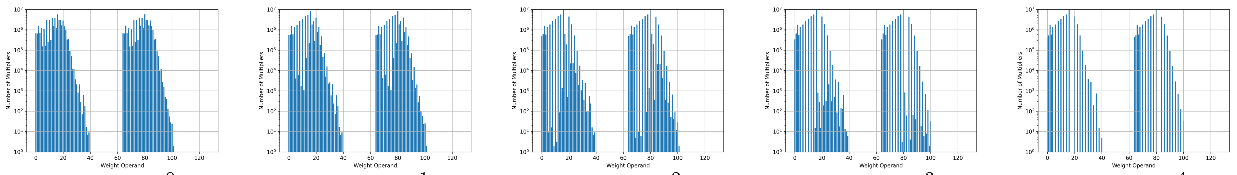
(a) $\lambda = 10^{-1}$ (b) $\lambda = 10^{0}$ (c) $\lambda = 10^{1}$ (d) $\lambda = 10^{2}$ (e) $\lambda = 10^{3}$

Figure 17. Quantized weight histogram for custom FP8 multiplier e4m3b7, with a regularization factor λ.



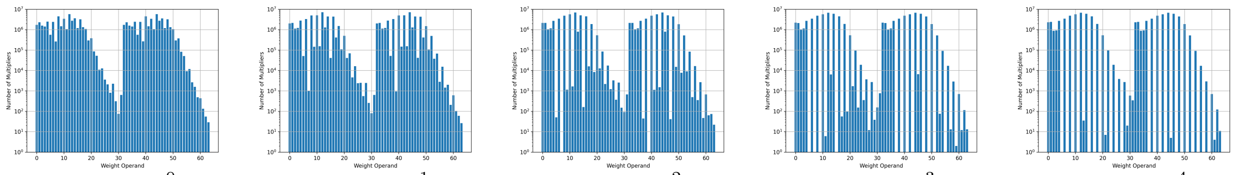
(a) $\lambda = 10^{-1}$ (b) $\lambda = 10^{0}$ (c) $\lambda = 10^{1}$ (d) $\lambda = 10^{2}$ (e) $\lambda = 10^{3}$

Figure 18. Quantized weight histogram for custom FP8 multiplier e5m2b15, with a regularization factor λ.



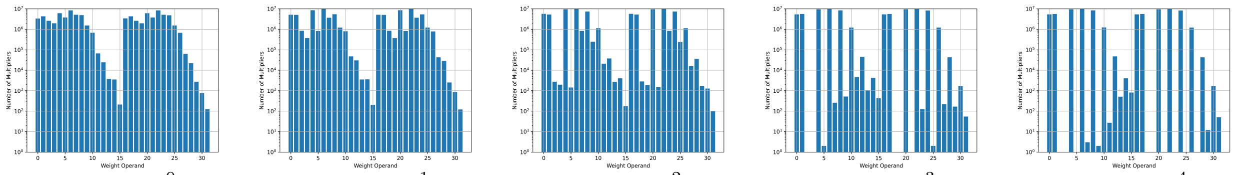
(a) $\lambda = 10^{0}$ (b) $\lambda = 10^{1}$ (c) $\lambda = 10^{2}$ (d) $\lambda = 10^{3}$ (e) $\lambda = 10^{4}$

Figure 19. Quantized weight histogram for custom FP7 multiplier e4m2b8, with a regularization factor λ.



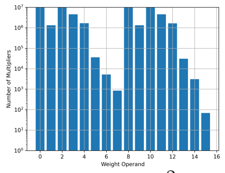
(a) $\lambda = 10^{0}$ (b) $\lambda = 10^{1}$ (c) $\lambda = 10^{2}$ (d) $\lambda = 10^{3}$ (e) $\lambda = 10^{4}$

Figure 20. Quantized weight histogram for custom FP6 multiplier e3m2b7, with a regularization factor λ.

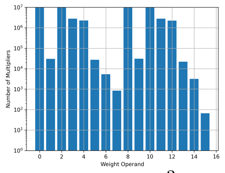


(a) $\lambda = 10^{0}$ (b) $\lambda = 10^{1}$ (c) $\lambda = 10^{2}$ (d) $\lambda = 10^{3}$ (e) $\lambda = 10^{4}$

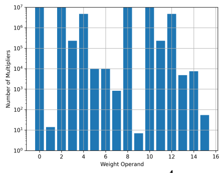
Figure 21. Quantized weight histogram for custom FP5 multiplier e3m1b7, with a regularization factor λ.



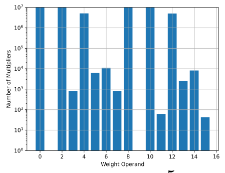
(a) $\lambda = 10^2$



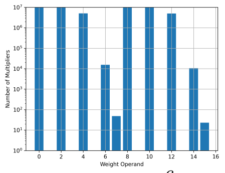
(b) $\lambda = 10^3$



(c) $\lambda = 10^4$

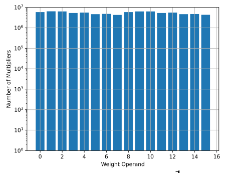


(d) $\lambda = 10^5$

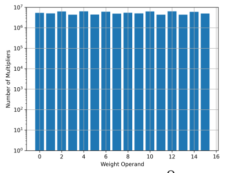


(e) $\lambda = 10^6$

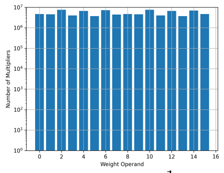
Figure 22. Quantized weight histogram for custom FP4 multiplier e3m0b6, with a regularization factor λ.



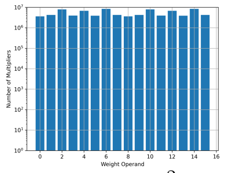
(a) $\lambda = 10^{-1}$



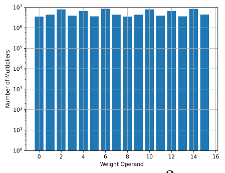
(b) $\lambda = 10^0$



(c) $\lambda = 10^1$

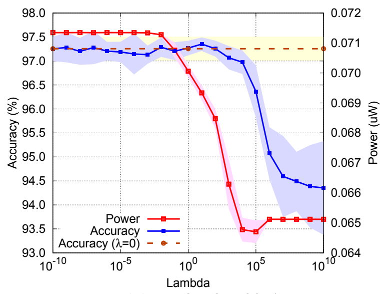


(d) $\lambda = 10^2$

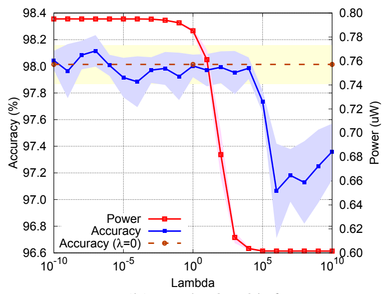


(e) $\lambda = 10^3$

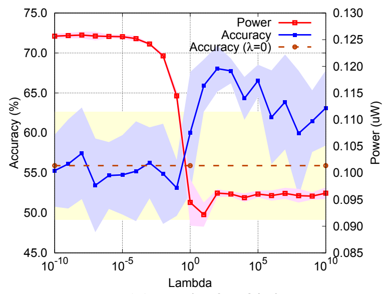
Figure 23. Quantized weight histogram for custom FP4 multiplier e0m3b4, with a regularization factor λ.



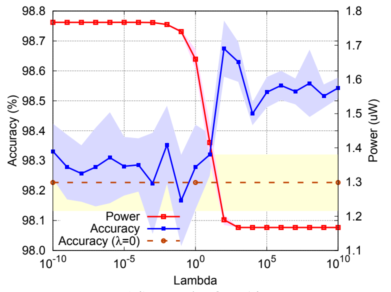
(a) FP3 e2m0b5



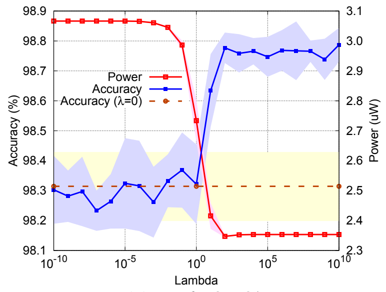
(b) FP4 e3m0b6



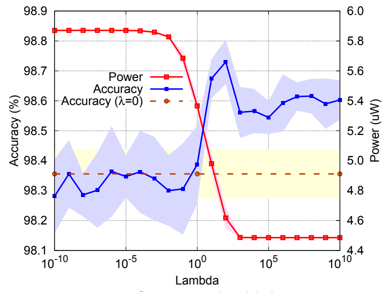
(c) FP4 e0m3b4



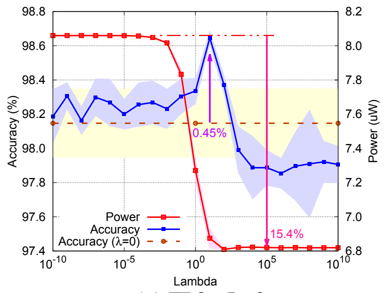
(d) FP5 e3m1b7



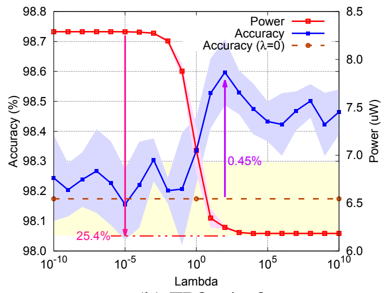
(e) FP6 e3m2b7



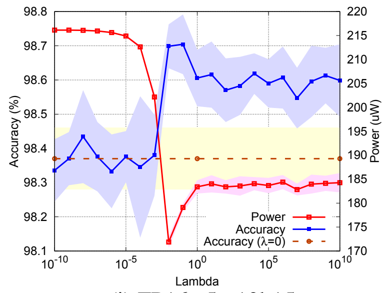
(f) FP7 e4m2b8



(g) FP8 e5m2



(h) FP8 e4m3



(i) FP16 e5m10b15

Figure 24. Power-aware quantization results across regularization factor λ.

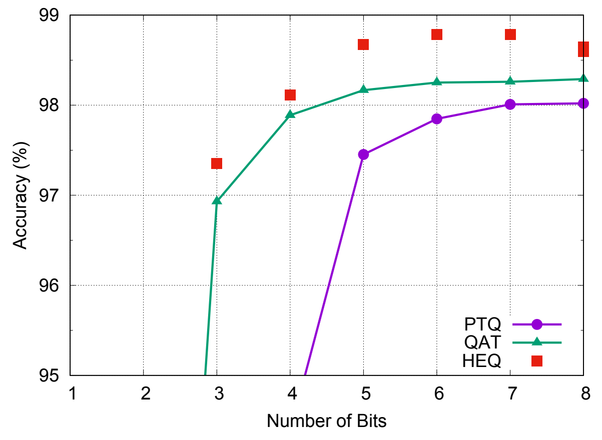
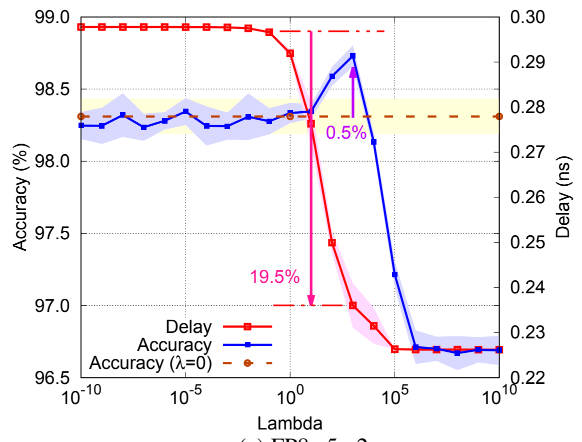
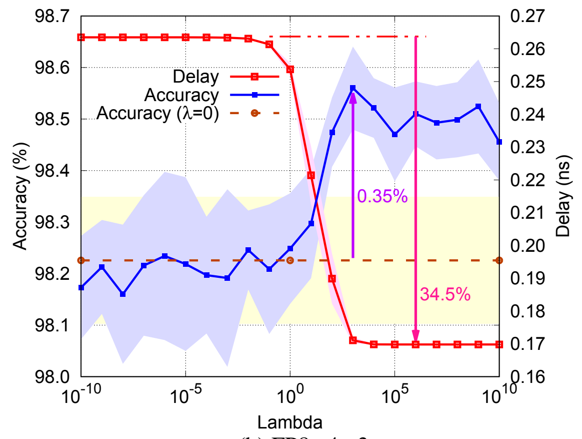


Figure 25. Accuracy vs. quantization bits.



(a) FP8 e5m2



(b) FP8 e4m3

Figure 27. Delay-aware quantization.

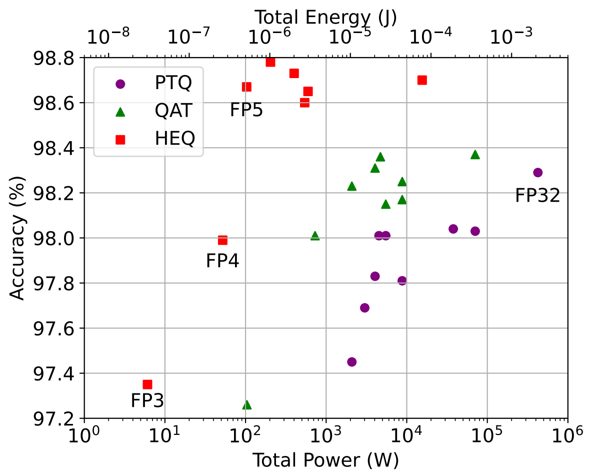


Figure 26. Accuracy vs. power tradeoff for foundation designs.