Reshape and Adapt for Output Quantization (RAOQ): Quantization-aware Training for In-memory Computing Systems

Bonan Zhang [1] **Chia-Yu Chen** [1 2] **Naveen Verma** [1 2]

Abstract

In-memory computing (IMC) has emerged as a promising solution to address both computation and data-movement challenges, by performing computation on data in-place directly in the memory array. IMC typically relies on analog operation, which makes analog-to-digital converters (ADCs) necessary, for converting results back to the digital domain. However, ADCs maintain computational efficiency by having limited precision, leading to substantial quantization errors in compute outputs. This work proposes RAOQ (Reshape and Adapt for Output Quantization) to overcome this issue, which comprises two classes of mechanisms including: 1) mitigating ADC quantization error by adjusting the statistics of activations and weights, through an activation-shifting approach (A-shift) and a weight reshaping technique (W-reshape); 2) adapting AI models to better tolerate ADC quantization through a bit augmentation method (BitAug), complemented by the introduction of ADC-LoRA, a low-rank approximation technique, to reduce the training overhead. RAOQ demonstrates consistently high performance across different scales and domains of neural network models for computer vision and natural language processing (NLP) tasks at various bit precisions, achieving state-of-the-art results with practical IMC implementations.

1. Introduction

Rapid advances in AI have greatly impacted various application domains, including computer vision, natural language processing, speech, etc. Recent generative AI breakthroughs have pushed the strength of AI even further, producing remarkably realistic and imaginative outputs, blurring the line between human- and machine-generated content (OpenAI, 2023; Team et al., 2023). However, increasing AI capability has come from increasing model complexity, with a sharp rise in both the number of compute operations and the number of parameters, placing huge demands on hardware resources (Villalobos et al., 2022; Smith et al., 2022).

This has driven the development of specialized hardware architectures to accelerate AI model computations. While digital accelerators have been widely deployed to improve compute efficiency, they do not address the large amount of data movement involved, which has been shown to pose a critical energy and performance bottleneck in state-of-the-art (SOTA) models (Verma et al., 2019). In-memory computing (IMC), on the other hand, performs computations in place on stored data, providing an approach to simultaneously address both compute efficiency and data movement.

Analog IMC enhances computational efficiency over digital methods, benefiting advanced AI models (Houshmand et al., 2023). However, a fundamental requirement of analog IMC is the need for analog-to-digital converters (ADCs), to provide compute outputs back to the digital domain. Importantly, ADCs introduce an additional source of quantization, which can substantially degrade accuracy in SOTA AI models. The level of such quantization error is fundamentally set by both the level of IMC parallelism and ADC resolution, which also directly sets the energy efficiency and throughput advantage. This is illustrated in Fig .1a, which compares the energy efficiency of IMC against digital accelerators, derived from the literature (Murmann; Lee et al., 2021; Jouppi et al., 2017). While IMC presents a significant energy efficiency advantage over digital accelerators, the advantage drops drastically as ADC precision is increased. Hence, it is critical to overcome such effects of ADC quantization to enable IMC efficiency in SOTA systems.

Unlike conventional quantizers, whose clipping/scaling parameters can be optimized during training, ADC quantization applies to analog compute results, where additional processing is not feasible, precluding the use of quantizers and the optimization degrees of freedom they offer. **In other words, ADC quantization incurs a fixed predefined quantization step and clipping values**. This key attribute distinguishes ADC quantization from traditional

[1]Princeton University, NJ, USA [2]EnCharge AI, CA, USA. Correspondence to: Bonan Zhang <bonanz@princeton.edu>.

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

quantization and imposes critical algorithmic challenges in IMC systems. Previous works introduce artificial clipping to ADC quantization at the hardware design stage (Gonugondla et al., 2020; Sakr & Shanbhag, 2021). However, this limits hardware flexibility in supporting various types of models, which may present different ADC-input data distributions and thus require different optimal clipping values.

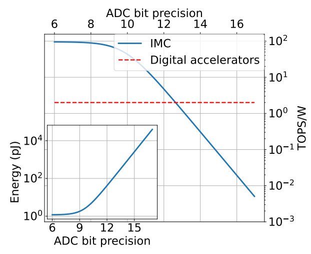
This paper presents RAOQ (Reshape and Adapt for Output Quantization), to tackle such quantization challenges at the algorithmic level. As neural networks generally are sensitive to drastic changes, we first perform quantization-aware training (QAT) for activations and weights only, and then apply RAOQ in another finetuning stage with ADC quantization introduced. We explore RAOQ across multiple applications, i.e., image classification on ImageNet (Deng et al., 2009), object detection on COCO 2017 (Lin et al., 2014), question answering on SQuAD 1.1 (Rajpurkar et al., 2016), and language modeling on WikiText-2/103 (Merity et al., 2016). This work is among the first to demonstrate approaches that enable IMC for inference across various scales of models and challenging datasets/tasks, particularly employing SOTA energy-efficiency design points for IMC. Our major contributions are as follows:

1. We conduct an analysis of the statistical attributes of activations and weights that yield a high signal-to-quantization-noise ratio (SQNR) in the presence of ADC quantization, and propose an activation-shifting method (A-shift) together with weight-reshaping techniques (W-reshape) to improve the SQNR.
2. We propose bit augmentation (BitAug), where the model is augmented in the dimension of ADC bit precision to aid the optimization process, assisting model adaptation to ADC quantization.
3. We propose ADC-LoRA, which adapts LoRA techniques (Hu et al., 2021) to address ADC quantization through an MSE-based initialization, achieving $> 45\times$ reduction of trainable parameters, which makes our proposed method applicable to large-scale AI models.
4. We perform extensive experiments on a wide range of models and tasks, including both CNN models (ResNet, MobileNetV2, YOLOv5) and transformer models (BERT, OPT, BLOOM), and across different quantization precisions. The consistently high performance achieved by our methods shows promise for their generalizability across challenging AI tasks.

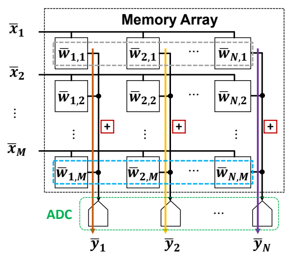
2. Background and Related Works

2.1. In-memory Computing (IMC)

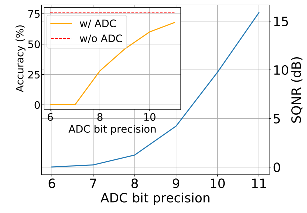
IMC aims to address both compute and data-movement costs in matrix-vector multiplications (MVMs), which are dominant operations in modern AI models. It stores matrix weights in a 2D array of memory bit cells as shown in Fig. 1b, and accesses compute results over multiple weight bits. This is achieved by performing multiplication in each bit cell between stored weight data and provided input data, and then accumulation to reduce the products in each column to a single result. The level of reduction, set by the row parallelism of IMC operation, thus determines the energy efficiency and throughput gains.



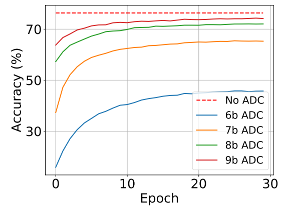
(a)



(b)



(c)



(d)

Figure 1. (a) Energy efficiency of IMC. (b) An illustration of an MVM operation via IMC. (c) SQNR and accuracy degradation due to ADC quantization. (d) Learning curves for vanilla QAT with ADC quantization involved.

IMC, realized via various memory technologies (e.g., SRAM, MRAM, etc.), can leverage analog operation for efficient computation, where the compute results then need to be converted back to the digital domain via ADCs (Lee et al., 2021; Yin et al., 2020; Deaville et al., 2022; Hsieh et al., 2023; Wan et al., 2022; Spetalnick et al., 2023). Such analog operation is sensitive to analog noise, which degrades the output signal-to-noise ratio (SNR). While methods have been proposed to overcome this issue (Zhang et al., 2022a; He et al., 2019; Rasch et al., 2023), they have only shown success at low efficiency IMC design points (low row parallelism, increased power) or on simple tasks. Instead, recent work has moved to a high-SNR form of IMC, overcoming analog noise sources while enabling higher row parallelism (Jia et al., 2022; Lee et al., 2021), but leaving ADC quantization as the primary challenge. As an example, Fig. 1c shows the degraded inference accuracy and signal-to-quantization-noise ratio (SQNR) due to ADC quantization. Consequently, such quantization now restricts the use of IMC in SOTA models and/or poses an ultimate limit on its efficiency and throughput. This work introduces efficient algorithmic approaches to address such a challenge, doing so without incurring additional hardware costs, to demonstrate applicability on a critical set of models.

2.2. Quantization-aware Training (QAT)

QAT restores model accuracy, which may otherwise degrade due to quantization noise, through a training process that adapts model parameters. QAT methods have been proposed to demonstrate SOTA accuracy in quantized networks (Jacob et al., 2017; Bhalgat et al., 2020; Choi et al., 2019; Wang et al., 2022; Esser et al., 2019). However, previous methods mainly focus on quantization with scale/clipping parameters that can be optimized during training. Such parameters are not available for ADC quantization, which applies to analog outputs from computation and where the quantization step is thus fixed. As a result, IMC shows substantially degraded accuracy with vanilla QAT[1], as seen in Fig. 1d.

To address ADC quantization in IMC, Jin et al. (2022) introduces a modified straight-through estimator (STE) (Bengio et al., 2013) along with calibration and rescaling techniques to assist QAT, demonstrating on CIFAR-10/100 datasets. Sun et al. (2021) proposes a non-uniform activation quantizer and a reduced quantization range, validating on CIFAR-10. Wei et al. (2020) proposes modified minmax quantizers for activations and weights to incorporate hardware statistics of IMC, testing on MNIST and CIFAR-10 datasets. While these prior works show success on relatively simple datasets, their success has not transferred to more complex datasets and AI tasks. In this work, we propose improved QAT techniques to enable SOTA accuracy applicable to various bit precisions on more challenging models and tasks.

2.3. Quantization with LoRA

Low-rank adaption (LoRA) reduces the trainable parameters, enabling efficient finetuning and reduction of memory requirements (Hu et al., 2021). It freezes the pre-trained weights and only optimizes a small number of parameters obtained via low-rank decomposition. Recent research has applied such a technique for neural network quantization (Dettmers et al., 2023; Li et al., 2023; Guo et al., 2023; Xu et al., 2023). A particular challenge is that the original initialization used in (Hu et al., 2021) does not translate well to quantization (Guo et al., 2023). Thus, initialization via SVD techniques has been proposed (Li et al., 2023). However, as we describe in Section 3.4, this approach does not apply to IMC ADC quantization. This work introduces an MSE-based approach to adapt LoRA seamlessly to our proposed algorithms for handling ADC quantization, providing a significant reduction of the training cost.

[1]Vanilla QAT refers to training with quantization effects present, without any additional proposed techniques applied.

3. RAOQ

This section begins with the problem formulation of ADC quantization. We then introduce the proposed RAOQ by describing each of its components. We use $\mathcal{L}_Q$ to denote the loss during the QAT stage and use $\mathcal{L}_A$ to denote the loss during the RAOQ stage [2].

3.1. ADC Quantization

As outlined in the previous sections, ADC quantization differs from traditional quantization issues, constrained by the design of the ADC itself and characterized by fixed clipping values/step sizes with inherently limited precision.

To formally define the IMC ADC quantization problem, let $x \in \mathbb{R}^M$ be a data vector of the activation X and let $w \in \mathbb{R}^M$ be a vector of an output channel of the weight W. Denote $\overline{x}$ and $\overline{w}$ as their quantized counterparts, an IMC column then computes a portion of MVM:

$$y = <\overline{x}, \overline{w}> = \sum_{i=1}^{M} \overline{w}_i \overline{x}_i \tag{1}$$

Note that convolutions can be converted to MVMs via *im2col* operations. For a b_x-bit activation, b_w-bit weight, b_a-bit ADC, and memory with dimension $M \times N$, assuming symmetric quantization is applied to weights, the ADC quantization and its quantization step Δ_a is defined as

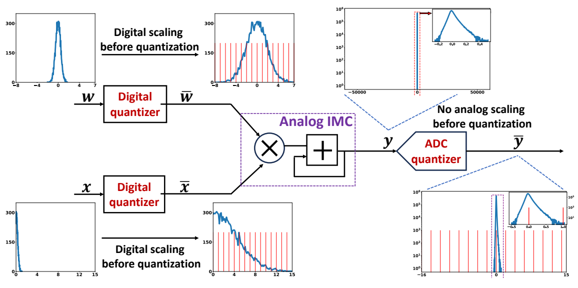
$$\overline{y} = \left\lfloor clip\left(\frac{y}{\Delta_a}, n_a, p_a\right)\right\rfloor \tag{2}$$

$$\Delta_a = \frac{2M(2^{b_x}-1)(2^{b_w-1}-1)}{2^{b_a}k} \tag{3}$$

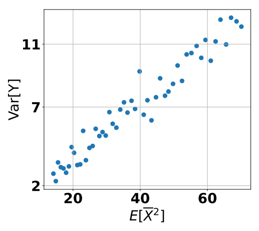
where $\lfloor\cdot\rfloor$ denotes the floor operation, and $clip(z, r_n, r_p)$ clips z to the range between r_n and r_p. Similar to conventional QAT, the gradient of the floor operation is approximated using STE (Bengio et al., 2013). Above, $(n_a, p_a) = (-2^{b_a-1}, 2^{b_a-1}-1)$. k **is a positive integer, serving as a hardware design parameter to provide fixed clipping (i.e., quantization step Δ_a is fixed)**, which makes such quantization particularly challenging. Eq. 3 assumes unsigned activations. For signed activations, we can replace $2^{b_x}-1$ with $2^{b_x-1}-1$. Fig. 2a shows the distribution of a typical ADC input compared to conventional quantizers. We see that the input concentrates around a small portion of the ADC range, resulting in a small signal, relative to its quantization step. A choice of large k could help to have a finer step Δ_a, but would potentially introduce substantial clipping error. As different layers and models lead to different statistics of the compute outputs (ADC inputs), there is no optimal Δ_a to rule them all. Thus, with no algorithmically controllable parameters for ADC quantization, the

[2]QAT stage refers to activation/weight quantization only, and RAOQ stage is with ADC quantization involved.

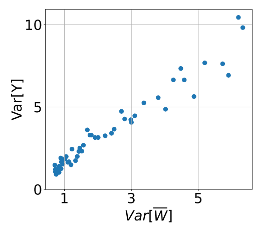
only degrees of freedom left are parameters applicable to the activations and weights.



(a)



(b)



(c)

Figure 2. (a) Illustration of conventional weight/activation quantization, where optimal digital scaling/clipping can be directly applied (red lines indicate quantization levels), and ADC quantization, where analog signals prevent the application of scaling/clipping. (b-c) Relationship of the variance of ADC input to $\mathbb{E}[\overline{X}^2]$ and $Var[\overline{W}]$.

3.2. SQNR Enhancement

Based on observations in Fig. 2a, we prefer the variance of the ADC input $Var\,[Y]$ to be maximized in order to maximize signal power and utilize as many ADC quantization levels as possible. This focus on 2^{nd}-order statistics, makes it natural to consider the dependence on the 2^{nd} moment of the activation X and weight W. Before the training starts, activations and weights are independent of each other, and $Var\,[Y]$ is proportionally set by $\mathbb{E}[\overline{X}^2]$ and $\mathbb{E}[\overline{W}^2]$. However, the assumption of independence generally does not hold after training, as activations and weights exhibit correlation through the neural network learning process. Nonetheless, we postulate that a more narrow relationship holds, namely that there is direct dependence between the 2^{nd} moments, and we conduct an empirical study to validate this. We randomly sample from widely used datasets (e.g., ImageNet), and also generate random input data. To manage computation complexity, we take the first few layers of various SOTA models to incorporate commonly used structures (e.g., convolution, attention, etc.) to perform QAT for this study. In Fig. 2b-2c, we plot the variance of the ADC input $Var[Y]$ vs. $\mathbb{E}[\overline{X}^2]$ and $\mathbb{E}[\overline{W}^2]$, respectively, and observe a proportional relationship. Further, since neural network weights are typically symmetrically distributed around zero (Bhalgat et al., 2020), $\mathbb{E}[\overline{W}^2]$ can be taken to be $Var[\overline{W}]$, and we postulate that $Var\,[Y]$ can be increased by maximizing $Var[\overline{W}]$ and $\mathbb{E}[\overline{X}^2]$, to improve IMC SQNR in the presence of ADC quantization. This rationale forms the basis of the A-shift and W-reshape as described below.

Activation shifting (A-shift). In order to maximize the 2^{nd} moment of the activation, it is desirable for activations to exhibit a concentration of mass at considerably large absolute values, i.e., distancing the mass from zero, so that the input distribution to the ADC has maximum variance. However, this is typically not the case with activations derived from functions like SiLU (Elfwing et al., 2018) and GELU (Hendrycks & Gimpel, 2016), which inherently exhibit significant mass distribution around small values in close proximity to zero, as shown in Fig. 3a (bottom left).

Exploiting the fact that quantizing these activations as a signed number or unsigned number does not have much impact on the overall performance (Bhalgat et al., 2020), we propose to treat them as unsigned numbers during quantization, and then convert them to signed numbers. This yields a distribution moved away from zero, to the advantage of ADC quantization. Such an unsigned-to-signed conversion can be implemented by a simple shift:

$$\overline{x}_s = \left\lfloor clip\left(\frac{x-\beta}{s_x}, 0, 2^{b_x}-1\right)\right\rceil - C = \overline{x} - C \quad (4)$$

where $\lfloor\cdot\rceil$ denotes round operation, β and s_x are learnable quantization parameters, C is the amount of shift. C is desired to be as large as possible for maximizing $\mathbb{E}[\overline{X}^2]$, but is limited by the minimum/maximum value that the quantized activation range can accommodate. Hence, C is chosen[3] to be 2^{b_x-1}. Fig. 3a (bottom) shows the A-shift process. We observe that the mass of $\overline{x}_s$ is concentrated at the most negative values, hence having an extremely large 2^{nd} moment. On the contrary, quantizing activations directly to a signed number prevents such a shift operation, resulting in a much smaller 2^{nd} moment. We compute the 2^{nd} moment numerically for the quantized activation from A-shift and from signed quantization based on Fig. 3a, ending up with 57.9 and 3.89, respectively. Our proposed approach produces a much greater 2^{nd} moment, roughly $15\times$ higher. Additionally, ReLU function naturally suits the A-shift approach, as they explicitly force the activations to be unsigned numbers. With such shifting, the IMC computation becomes

$$y = \sum_{i=1}^{M} \overline{w}_i \overline{x}_i = \sum_{i=1}^{M} \overline{w}_i \overline{x}_{s,i} + \underbrace{C\,\overline{w}_i}_{\text{offset}} \quad (5)$$

The additional offset introduced by A-shift can be precomputed offline and thus does not add any overhead when

[3] The range of a signed number is from -2^{b_x-1} to $2^{b_x-1}-1$

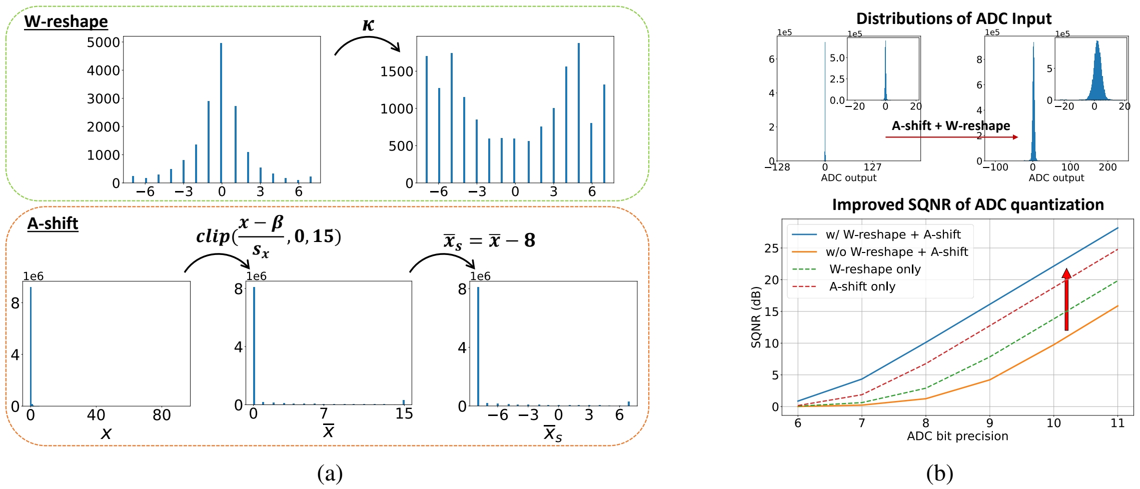


Figure 3. (a) Demonstration of W-reshape and A-shift for 4b weights and activations. (b) SQNR improvement under ADC quantization.

performing inference on IMC systems. The applicability of A-shift on IMC with other number representations is described in Appendix B.

Weight reshaping (W-reshape). To maximize $Var[\overline{W}]$, one option is to perform aggressive scaling during quantization. However, this is expected to introduce substantial clipping error, posing an adverse trade-off with weight distortion. Thus, we seek an alternate approach to increasing $Var[\overline{W}]$, by adjusting the distribution shape.

Neural network weights typically exhibit a symmetric distribution in the exponential family, e.g., Gaussian or Laplace distribution (Banner et al., 2019), which results in relatively low variance. Thus, we leverage an observation from previous work that a kurtosis penalty, which describes the tailedness of a distribution, on the weight can be used to flatten its distribution (Shkolnik et al., 2020). Based on our findings in Fig. 2c, we propose to apply the kurtosis directly on $\overline{W}$ and remove the constraint of matching a uniform distribution originally used in (Shkolnik et al., 2020), to push $\overline{W}$ further towards the endpoints. Such penalty is defined as

$$\kappa = \mathbb{E}\left[\left(\frac{\overline{W} - \mu_{\overline{W}}}{\sigma_{\overline{W}}}\right)^4\right] \tag{6}$$

where $\mu_{\overline{W}}$ and $\sigma_{\overline{W}}$ denote the mean and standard deviation of $\overline{W}$. The final loss, combined with the original loss function $\mathcal{L}_c$ against the ground truth during QAT is

$$\mathcal{L}_Q = \mathcal{L}_c + \lambda_\kappa \sum_l \kappa_l \tag{7}$$

where λ_κ is a coefficient to balance different loss terms, and l is an index for neural network layers. We provide an analysis of the selection of λ_κ in Appendix A. Fig. 3a (top) shows a comparison between the quantized weight without and with incorporating κ. We see that the proposed method successfully reshapes the weight distribution to have a much larger variance, i.e., 4× more than the case without κ.

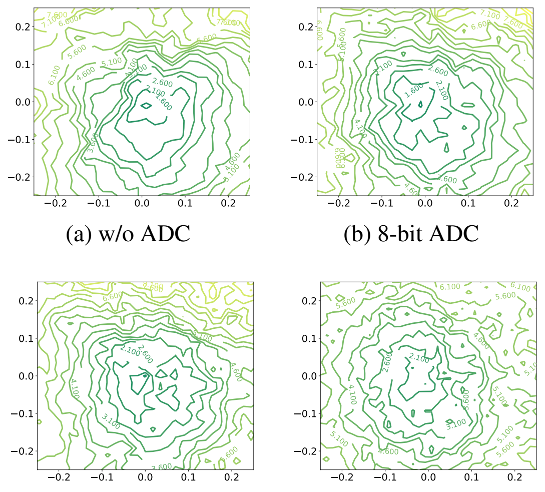
Impact of W-reshape and A-shift Fig. 3b summarizes the results obtained by applying W-reshape and A-shift. We can visually observe an increase in the variance of the ADC input and an improved SQNR accordingly.

3.3. SQNR Adaptation for Neural Networks

Quantization fundamentally sacrifices information in exchange for model compression. While SQNR is improved through Eq. 4 - Eq. 7, quantization imposed by the ADC is observed to make SGD-based optimization more challenging due to its fixed quantization step and limited precision. Fig. 4 shows the loss surfaces of MobileNetV2 in two randomly selected parameter dimensions for visualization (Li et al., 2018). As seen, ADC quantization causes a less smooth surface with additional local minima. These attributes reduce the likelihood of arriving at preferred (low-loss) minima during the training process. Approaches are thus required to adapt the model to this extra quantization.

Bit augmentation (BitAug). We seek an approach that facilitates a greater volume of information to be backpropagated so that the model parameters can be optimized more effectively. Inspired by NetAug (Cai et al., 2021) where a tiny model is inserted into larger models during training, we augment the network with ADCs of different bit precisions. At each iteration, we first pass the desired ADC bit to the model and then pass other bit precisions from a pre-defined set $\mathbb{B}$ to the model. The general form of BitAug is

$$\mathcal{L}_A = \mathcal{L}(\theta, b_a) + \lambda_b \sum_{i=1}^{B} \mathcal{L}(\theta, b_{a,i}) \tag{8}$$



(a) w/o ADC (b) 8-bit ADC

(c) 7-bit ADC (d) 6-bit ADC

Figure 4. Loss surfaces with A-shift and W-reshape applied for 4-bit activations and weights.

where θ denotes network parameters, λ_b is the coefficient of the BitAug loss, B is the size of $\mathbb{B}$, and $b_{a,i}$ is a sample from $\mathbb{B}$. Elements in $\mathbb{B}$ are chosen to be neighbors of the target ADC bit precision. Given the complexity of optimization with ADC quantization, we employ the assistance of other bit precisions. The information associated with the various ADC bit precisions is then represented in their respective gradients, which get accumulated during the backward path for more optimal updating of model parameters, i.e.,

$$\theta^{t+1} = \theta^t - \eta \frac{\partial \mathcal{L}(\theta^t, b_a)}{\partial \theta^t} - \eta \lambda_b \sum_{i=1}^{B} \frac{\partial \mathcal{L}(\theta^t, b_{a,i})}{\partial \theta^t} \tag{9}$$

where t indicates the current training step, η denotes the learning rate. Following the insights from Fig. 4, our study suggests that BitAug facilitates the learning process by mitigating getting stuck at local minima. We include our analysis details in Appendix C.

However, such an aggregation of multiple augmented models is computationally expensive. Following a similar strategy as (Cai et al., 2021), we randomly sample an ADC bit precision for each iteration, i.e.,

$$\mathcal{L}_A = \mathcal{L}(\theta, b_a) + \lambda_b \mathcal{L}(\theta, \widetilde{b}_a) \tag{10}$$

where $\widetilde{b}_a$ is a uniformly sampled bit precision from $\mathbb{B}$. We observe that doing this not only improves the computational efficiency by a factor of B, but achieves better performance than running all ADC bits simultaneously. Additionally, the selection of $\mathbb{B}$ is also critical. For instance, if we only sample lower precision ADC, we are essentially adding noise to the training process, which causes performance degradation. We include a quantitative study in Appendix D.

3.4. Training Overhead Reduction

Although the proposed methods are effective for handling ADC quantization, they introduce additional training overhead. While the overhead from A-shift and W-reshape is negligible, BitAug requires $\geq 14\%$ more GPU memory due to the gradient accumulation of different ADC bits. This is particularly an issue when the model size is scaled up.

To alleviate such training costs, we propose ADC-LoRA that makes use of LoRA (Hu et al., 2021) techniques to reduce the number of trainable parameters, thus reducing the amount of gradient accumulation. Let Q_w, Q_x be the quantizers of weights and activations, respectively, and Q_A be the quantizer of ADC defined in Eq. 2. Given a pre-trained weight $W \in \mathbb{R}^{m \times n}$, the ADC-quantized compute output with LoRA applied has the form

$$\overline{Y} = Q_A(Q_x(X) Q_w(W + AB)) \tag{11}$$

where $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{r \times n}$ are learnable low-rank matrices with $r \ll min(m, n)$. By keeping W fixed, the number of trainable parameters gets dramatically reduced as shown in Table 4.

While the SVD-based approach has been shown as an effective initialization of LoRA for weight quantization (Li et al., 2023; Guo et al., 2023), it does not apply to ADC quantization due to the extra quantization (i.e., ADC quantization) operated on LoRA parameters as indicated in Eq. 11. A more detailed discussion and comparison of different methods are provided in Appendix E. Hence we propose to initialize and warmup these parameters via an MSE optimization:

$$\arg\min_{A,B} \| Q_x(X) Q_w(W) - \overline{Y} \|_F^2 \tag{12}$$

where $\overline{Y}$ is defined in Eq. 11, and $\|\cdot\|_F$ denotes Frobenious norm. This can be solved via gradient-based methods by employing STE for the quantizers. We include a summary of the overall RAOQ framework in Appendix I.

4. Experiments

We consider a general IMC architecture as shown in Fig. 1b. While exploring different IMC architectures is not the focus of this paper, we include experiments on the impact of RAOQ with various IMC configurations in Appendix F for interested readers. In this section, we focus on an IMC system with aggressive memory dimensions of 512×512, taking 4-bit inputs and processing 4-bit weights, with ADCs having $k = 4$. Higher-precision activations and weights are mapped to the IMC via matrix tiling.

The proposed methods are evaluated on both CNN and transformer models for various AI tasks: image classification on

Table 1. Evaluation accuracy of RAOQ on CNN models with various ADC bit precisions.

Model	Dataset	Full precision	b_x, b_w	No ADC	$b_a = 7$ QAT*	$b_a = 7$ RAOQ	$b_a = 8$ QAT*	$b_a = 8$ RAOQ	$b_a = 9$ QAT*	$b_a = 9$ RAOQ
ResNet18	ImageNet	69.76	8,8	70.66	60.12	**70.28**	66.03	**70.46**	66.65	**70.60**
			4,4	70.49	59.42	**70.23**	65.71	**70.45**	66.61	**70.49**
ResNet50	ImageNet	76.23	8,8	76.53	65.47	**76.25**	73.83	**76.46**	75.01	**76.51**
			4,4	76.31	65.25	**76.15**	72.05	**76.27**	74.16	**76.32**
MobileNetV2	ImageNet	71.81	8,8	71.89	62.09	**71.57**	66.72	**71.79**	69.13	**71.93**
			4,4	70.47	61.51	**70.22**	66.67	**70.46**	68.55	**70.45**
EfficientNet-lite0	ImageNet	75.12	8,8	74.31	61.27	**73.58**	68.11	**74.08**	68.85	**74.21**
			4,4	72.84	61.21	**72.18**	67.03	**72.76**	67.85	**72.82**
YOLOv5s	COCO2017	37.20$^\diamond$	8,8	36.60	1.30	**34.73**	8.02	**35.82**	24.03	**36.41**
			4,4	33.78	10.13	**32.23**	20.32	**33.49**	28.49	**33.89**

*: Vanilla QAT. $^\diamond$: Result trained by ourselves in FP32 rather than original mixed-precision.

Table 2. Evaluation of RAOQ on transformer models with various ADC bit precisions for 4-bit activations and weights. Encoder (BERT) and decoder (OPT, BLOOM) models are measured in F1 score (larger is better) and perplexity (smaller is better), respectively.

Model	Dataset	Full precision	No ADC	$b_a = 7$ QAT*	$b_a = 7$ RAOQ	$b_a = 8$ QAT*	$b_a = 8$ RAOQ	$b_a = 9$ QAT*	$b_a = 9$ RAOQ
BERT-base	SQuAD 1.1 ↑	88.58	87.75	64.46	**87.31**	82.43	**87.67**	84.53	**87.75**
BERT-large	SQuAD 1.1 ↑	91.00	89.57	64.02	**88.89**	79.52	**89.28**	84.10	**89.52**
OPT-125M	WikiText-2 ↓	20.21	23.60	441.64	**26.15**	69.78	**24.71**	35.38	**23.82**
	WikiText-103 ↓	16.15	20.03	378.86	**22.24**	60.54	**21.08**	34.65	**20.38**
OPT-350M	WikiText-2 ↓	17.80	19.17	621.12	**22.07**	384.95	**19.88**	27.15	**19.45**
	WikiText-103 ↓	15.09	17.56	590.28	**19.78**	110.36	**18.34**	27.56	**17.89**
OPT-1.3B	WikiText-2 ↓	14.02	14.24	557.36	**15.82**	465.75	**14.42**	30.14	**14.30**
	WikiText-103 ↓	12.08	12.40	595.65	**13.33**	552.11	**12.68**	28.12	**12.48**
BLOOM-560M	WikiText-2 ↓	22.38	23.87	464.31	**26.53**	77.64	**24.94**	32.55	**24.16**
	WikiText-103 ↓	16.52	18.62	425.08	**20.88**	69.26	**19.20**	30.02	**18.95**
BLOOM-1.1B	WikiText-2 ↓	18.69	19.21	510.63	**22.05**	48.86	**20.06**	28.12	**19.84**
	WikiText-103 ↓	14.89	16.14	460.14	**17.78**	45.24	**16.77**	27.98	**16.45**
BLOOM-1.7B	WikiText-2 ↓	15.44	16.75	852.65	**18.26**	387.98	**17.06**	26.76	**16.88**
	WikiText-103 ↓	13.64	14.28	743.68	**15.85**	208.56	**14.74**	26.10	**14.44**

*: Vanilla QAT. ↑: larger results indicate better performance. ↓: smaller results indicate better performance.

ImageNet (Deng et al., 2009) with ResNet18/50 (He et al., 2016), MobileNetV2 (Sandler et al., 2018), and EfficientNet-lite0 (Tan & Le, 2019); object detection on COCO 2017 (Lin et al., 2014) with YOLOv5s (Jocher et al., 2022); question-answering on SQuAD 1.1 (Rajpurkar et al., 2016) with BERT-base/large (Devlin et al., 2018); language modeling on WikiText-2 and WikiText-103 (Merity et al., 2016) with OPT-125M/350M/1.3B (Zhang et al., 2022b) and BLOOM-560M/1.1B/1.7B (Workshop et al., 2022). The first and last layers are kept in 8-bit for fidelity reasons. LoRA techniques are applied on models with $>$ 200M parameters to maintain reasonable memory usage. We start from pre-trained full precision models, and first perform QAT based on LSQ+ (Bhalgat et al., 2020) for activations and weights with the proposed W-reshape and A-shift methods. The choice of the QAT quantizer does not affect our conclusion, and we include validation in Appendix G. We then add ADC quantization along with other RAOQ techniques for another stage of finetuning. We treat QAT results without ADC quantization as our baseline. All of them match SOTA results. For a fair comparison, we also perform vanilla QAT (i.e., without any proposed methods involved) for ADC quantization. We sweep the ADC bit precision to demonstrate the robustness and generality of our approaches. Experiments are performed on Nvidia A100 GPUs. Further training details are described in Appendix J.

4.1. Main Results

CNN models. Table 1 summarizes the results for 4-bit and

Table 3. Comparison of different methods for ADC quantization on CIFAR-10. M denotes the memory inner-dimension, and the column IMC indicates accuracy under ADC quantization.

MODEL	METHOD	b_x, b_w, b_a	M	FULL PRECISION	NO ADC	IMC	DEGRADATION
RESNET20	(JIN ET AL., 2022)	4,4,7	9	–	91.60	91.00	-0.60[b]
		4,4,3	9			81.70	-9.30[b]
	RAOQ	4,4,7	9	92.32	92.23	**92.32**	+0.09[b]
		4,4,3	9			**89.34**	-2.89[b]
RESNET18[a]	(SUN ET AL., 2021)	4,4,4	256	88.87	–	86.55	-2.32[c]
	RAOQ	4,4,4	256	92.10	92.13	**90.48**	-1.65[c]
RESNET18	(WEI ET AL., 2020)	2,2,4	9	92.01	89.62	83.37	-6.25[b]
		2,2,4	36			87.56	-2.06[b]
	RAOQ	2,2,4	9	93.21	92.26	**91.90**	-0.36[b]
		2,2,4	36			**91.81**	-0.45[b]

[a] Channels are reduced to 1/4 of the original ResNet18. [b] Accuracy drop of IMC ADC quantization with respect to no-ADC case. [c] Accuracy drop with respect to full precision.

8-bit activations and weights. As seen, the proposed RAOQ successfully preserves the results to the baseline level and significantly outperforms vanilla QAT in all cases, with $< 1\%$ accuracy drop for image classification and $< 2\%$ degradation in mAP for object detection.

Transformer models. Table 2 shows the results for 4-bit activations and weights. Encoder models (BERT) exhibit $< 1\%$ degradation in F1 score across multiple ADC bits. Decoder models (OPT, BLOOM) are evaluated using perplexity, a metric known to be sensitive to quantization (Yao et al., 2022). As seen, RAOQ restores the results to the baseline level, which degrades significantly using only vanilla QAT. For instance, OPT-1.3B sees a < 1.6 perplexity drop using RAOQ, compared to a > 15 drop with vanilla QAT.

4.2. Comparison with Other IMC QAT Methods

As mentioned, previous works focus on ADC quantization in IMC on small datasets. Table 3 shows a comparison of our proposed RAOQ approach with other works on CIFAR-10. These works are based on various memory technologies (e.g., SRAM, ReRAM). For a fair comparison, we construct the same model, following the same configurations as these works (e.g., bit precisions, memory dimensions, applicable hardware noise levels). We see that RAOQ outperforms all other methods, leading to much less degradation regardless of IMC technology and configurations. We provide additional comparisons on more complex tasks and datasets, with prior works implemented by ourselves, shown in Appendix H for interested readers.

4.3. Effectiveness of ADC-LoRA

To demonstrate the impact of ADC-LoRA on our training overhead, we show the reduction of trainable parameters as summarized in Table 4. To obtain results shown in Table 2, we choose rank 8 for BERT-large and rank 32 for OPT-1.3B and BLOOM 1.1B, implementing LoRA adaptors in both attention layers and feed-forward layers. As seen, such configuration allows us to achieve $> 45\times$ reduction of the training parameters, which potentially enables IMC systems to handle larger AI models with the assistance of the proposed RAOQ.

Table 4. Impact of ADC-LoRA on trainable parameters.

	BERT-LARGE	OPT-1.3B	BLOOM 1.1B
FULL SIZE	345M	1.3B	1.1B
ADC-LORA	3.5M	28M	13M

4.4. Ablation Study

We investigate the impact of each proposed technique in RAOQ. In particular, we use BERT-base and ResNet50 with 4-bit activations and weights, and 8-bit ADCs for the study. The results are summarized in Table 5. The first row corresponds to the results of vanilla QAT. Each check mark indicates the presence of a specific technique. As seen, all of the proposed techniques improve the degraded performance due to ADC quantization. Comparatively, A-shift and BitAug exhibit more significant impacts on the network performance, one contributing to boosting SQNR and the other responsible for model optimization.

Furthermore, we evaluate the isolated impact of ADC-LoRA on the model accuracy. Table 6 shows a comparison of models with and without ADC-LoRA applied. Similar to our experiments above, results for BERT-large are measured in F1 score (higher is better), while OPT 1.3B and BLOOM 1.7B are measured in perplexity (lower is better). The eval-

Table 5. Impact of different methods. The check mark indicates the use of the corresponding method.

A-SHIFT	W-RESHAPE	BITAUG	BERT-BASE	RESNET50
			82.43	72.05
✓			84.24	75.77
	✓		83.06	75.01
		✓	85.10	75.65
✓	✓		86.12	76.02
✓	✓	✓	87.67	76.27

uation is based on 4-bit activations and weights, and 8-bit ADCs. As seen, our proposed RAOQ approaches successfully achieve high performance in various models regardless of the application of ADC-LoRA. On the other hand, our proposed ADC-LoRA is able to preserve such high performance with a reduced training overhead.

Table 6. Isolated impact of ADC-LoRA on the model accuracy.

	BERT-LARGE	OPT-1.3B	BLOOM-1.7B
NO ADC	89.57	14.24	16.75
NO ADC-LORA	89.08	14.39	17.05
ADC-LORA	89.28	14.42	17.06

5. Relation to Analog Noise

Although this work focuses on ADC quantization, we also explore the impact of analog noise that is fundamentally present, to complete our work. Techniques for addressing analog noise have been well studied as mentioned in Section 2.1, which can potentially be integrated with our proposed RAOQ approach. Table 7 shows examples of such an integration to tackle both ADC quantization and analog noise with 4-bit weights and activations, and 8-bit ADCs. We apply simple additive noise on the weights and outputs with statistics from (Long et al., 2019) and (Lee et al., 2021), respectively. We see that < 0.1 degradation is observed even with analog noise present. Nevertheless, analog noise can vary depending on specific memory technologies and topologies and can have complex distributions. We leave this for future study.

Table 7. Impact of analog noise on RAOQ.

	RESNET50	MOBILENETV2	BERT-BASE
NOISE FREE	76.27	70.46	87.67
(LEE ET AL., 2021)	76.23	70.44	87.62
(LONG ET AL., 2019)	76.26	70.44	87.64

6. Conclusion

Analog IMC has shown substantial promise to simultaneously enhance compute efficiency and data-movement costs for AI inference. However, the associated ADC quantization restricts the performance of SOTA models applied to challenging tasks. This work proposes RAOQ to tackle such quantization. Specifically, we propose W-reshape and A-shift, to maximize SQNR following ADC quantization via adjusting the statistics of weights and activations. We further propose BitAug to improve the optimization process and introduce ADC-LoRA to mitigate the training overhead. Our work has been evaluated on various datasets, models, and bit precisions, achieving consistently high performance. The generalizability and robustness of RAOQ demonstrate the feasibility of applying IMC to challenging AI tasks.

Acknowledgements

This work is funded in part by the Defense Advanced Research Projects Agency (DARPA) under the OPTIMA program, agreement no. HR00112490300.

Impact Statement

IMC presents one of the most promising pathways to substantially improve AI compute efficiency, towards addressing the ultimate sustainability of AI applications adopting increasingly large and complex models. This work addresses the most critical current limiter of SOTA IMC, which is the efficiency-vs.-SNR tradeoff set by ADC quantization. The new algorithmic approaches presented for handling ADC quantization enable IMC application, without degrading efficiency, to more advanced models than previously reported, thereby advancing the pathway to IMC adoption for the most compute intensive AI uses of interest.

References

Banner, R., Nahshan, Y., and Soudry, D. *Post Training 4-Bit Quantization of Convolutional Networks for Rapid-Deployment*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Bengio, Y., Léonard, N., and Courville, A. C. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013.

Bhalgat, Y., Lee, J., Nagel, M., Blankevoort, T., and Kwak, N. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 696–697, 2020.

Cai, H., Gan, C., Lin, J., and Han, S. Network augmentation for tiny deep learning. *arXiv preprint arXiv:2110.08890*, 2021.

Choi, J., Venkataramani, S., Srinivasan, V., Gopalakrishnan, K., Wang, Z., and Chuang, P. I.-J. Accurate and efficient 2-bit quantized neural networks. In *Conference on Machine Learning and Systems*, 2019. URL `https://api.semanticscholar.org/CorpusID:96438794`.

Deaville, P., Zhang, B., and Verma, N. A 22nm 128-kb mram row/column-parallel in-memory computing macro with memory-resistance boosting and multi-column adc readout. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pp. 268–269, 2022. doi: 10.1109/VLSITechnologyandCir46769.2022.9830153.

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.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dong, Q., Sinangil, M. E., Erbagci, B., Sun, D., Khwa, W.-S., Liao, H.-J., Wang, Y., and Chang, J. A 351tops/w and 372.4gops compute-in-memory sram macro in 7nm finfet cmos for machine-learning applications. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 242–244, 2020. doi: 10.1109/ISSCC19947.2020.9062985.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., and Modha, D. S. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

Gonugondla, S. K., Sakr, C., Dbouk, H., and Shanbhag, N. R. Fundamental limits on energy-delay-accuracy of in-memory architectures in inference applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41:3188–3201, 2020.

Guo, H., Greengard, P., Xing, E. P., and Kim, Y. Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning. *arXiv preprint arXiv:2311.12023*, 2023.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, Z., Lin, J., Ewetz, R., Yuan, J.-S., and Fan, D. Noise injection adaption: End-to-end reram crossbar non-ideal effect adaption for neural network mapping. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2019.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Houshmand, P., Sun, J., and Verhelst, M. Benchmarking and modeling of analog and digital sram in-memory computing architectures. *arXiv preprint arXiv:2305.18335*, 2023.

Hsieh, S.-E., Wei, C.-H., Xue, C.-X., Lin, H.-W., Tu, W.-H., Chang, E.-J., Yang, K.-T., Chen, P.-H., Liao, W.-N., Low, L. L., Lee, C.-D., Lu, A.-C., Liang, J., Cheng, C.-C., and Kang, T.-H. 7.6 a 70.85-86.27tops/w pvt-insensitive 8b word-wise acim with post-processing relaxation. In *2023 IEEE International Solid- State Circuits Conference (ISSCC)*, pp. 136–138, 2023. doi: 10.1109/ISSCC42615.2023.10067335.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A. G., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2017.

Jia, H., Ozatay, M., Tang, Y., Valavi, H., Pathak, R., Lee, J., and Verma, N. Scalable and programmable neural network inference accelerator based on in-memory computing. *IEEE Journal of Solid-State Circuits*, 57(1):198–211, 2022. doi: 10.1109/JSSC.2021.3119018.

Jin, Q., Chen, Z., Ren, J., Li, Y., Wang, Y., and Yang, K. Pim-qat: Neural network quantization for processing-in-memory (pim) systems. *arXiv preprint arXiv:2209.08617*, 2022.

Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., Michael, K., TaoXie, Fang, J., imyhxy, Lorna, Zeng, Y., Wong, C., V, A., Montes, D., Wang, Z., Fati, C., Nadar, J., Laughing, UnglvKitDe, Sonck, V., tkianai, yxNONG, Skalski, P., Hogan, A., Nair, D., Strobel, M., and Jain, M. ultralytics/yolov5:

v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, November 2022. URL https://doi.org/10.5281/zenodo.7347926.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

Lee, J., Valavi, H., Tang, Y., and Verma, N. Fully row/column-parallel in-memory computing sram macro employing capacitor-based mixed-signal computation with 5-b inputs. In *2021 Symposium on VLSI Circuits*, pp. 1–2, 2021. doi: 10.23919/VLSICircuits52068.2021.9492444.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

Li, Y., Yu, Y., Liang, C., He, P., Karampatziakis, N., Chen, W., and Zhao, T. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014.

Long, Y., She, X., and Mukhopadhyay, S. Design of reliable dnn accelerator with un-reliable reram. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1769–1774, 2019. doi: 10.23919/DATE.2019.8715178.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Murmann, B. ADC Performance Survey 1997-2023. [Online]. Available: https://github.com/bmurmann/ADC-survey.

OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Rasch, M. J., Mackin, C., Gallo, M. L., Chen, A., Fasoli, A., Odermatt, F., Li, N., Nandakumar, S. R., Narayanan, P., Tsai, H., Burr, G. W., Sebastian, A., and Narayanan, V. Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators. *Nature Communications*, 14, 2023. URL https://api.semanticscholar.org/CorpusID:256900830.

Sakr, C. and Shanbhag, N. R. Signal processing methods to enhance the energy efficiency of in-memory computing architectures. *IEEE Transactions on Signal Processing*, 69:6462–6472, 2021. doi: 10.1109/TSP.2021.3130488.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Shkolnik, M., Chmiel, B., Banner, R., Shomron, G., Nahshan, Y., Bronstein, A. M., and Weiser, U. C. Robust quantization: One model to rule them all. *ArXiv*, abs/2002.07686, 2020.

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

Spetalnick, S. D., Chang, M., Konno, S., Crafton, B., Lele, A. S., Khwa, W.-S., Chih, Y.-D., Chang, M.-F., and Raychowdhury, A. A 2.38 mcells/mm2 9.81 -350 tops/w rram compute-in-memory macro in 40nm cmos with hybrid offset/ioff cancellation and icell rblsl drop mitigation. In *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pp. 1–2, 2023. doi: 10.23919/VLSITechnologyandCir57934.2023.10185424.

Sun, H., Zhu, Z., Cai, Y., Zeng, S., Qiu, K., Wang, Y., and Yang, H. Reliability-aware training and performance modeling for processing-in-memory systems. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 847–852, 2021.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Verma, N., Jia, H., Valavi, H., Tang, Y., Ozatay, M., Chen, L.-Y., Zhang, B., and Deaville, P. In-memory computing: Advances and prospects. *IEEE Solid-State Circuits Magazine*, 11(3):43–55, 2019. doi: 10.1109/MSSC.2019.2922889.

Villalobos, P., Sevilla, J., Besiroglu, T., Heim, L., Ho, A., and Hobbhahn, M. Machine learning model sizes and the parameter gap. *arXiv preprint arXiv:2207.02852*, 2022.

Wan, W., Kubendran, R., Schaefer, C., Eryilmaz, S. B., Zhang, W., Wu, D., Deiss, S., Raina, P., Qian, H., Gao, B., et al. A compute-in-memory chip based on resistive random-access memory. *Nature*, 608(7923):504–512, 2022.

Wang, N., Liu, C.-C. C., Venkataramani, S., Sen, S., Chen, C.-Y., El Maghraoui, K., Srinivasan, V. V., and Chang, L. Deep compression of pre-trained transformer models. *Advances in Neural Information Processing Systems*, 35: 14140–14154, 2022.

Wei, W.-C., Jhang, C.-J., Chen, Y.-R., Xue, C.-X., Sie, S.-H., Lee, J.-L., Kuo, H.-W., Lu, C.-C., Chang, M.-F., and Tang, K.-T. A relaxed quantization training method for hardware limitations of resistive random access memory (reram)-based computing-in-memory. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6:45–52, 2020.

Workshop, B., Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Xu, Y., Xie, L., Gu, X., Chen, X., Chang, H., Zhang, H., Chen, Z., Zhang, X., and Tian, Q. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*, 2023.

Yao, Z., Yazdani Aminabadi, R., Zhang, M., Wu, X., Li, C., and He, Y. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35: 27168–27183, 2022.

Yin, S., Jiang, Z., Seo, J.-S., and Seok, M. Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits*, 55 (6):1733–1743, 2020. doi: 10.1109/JSSC.2019.2963616.

Zhang, B., Deaville, P., and Verma, N. Statistical computing framework and demonstration for in-memory computing systems. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, pp. 979–984, 2022a. doi: 10.1145/3489517.3530557.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022b.

A. Impact of W-reshape on Inference Accuracy

In Section 3.2 of the paper, we introduce a weight reshaping method (W-reshape) to adjust weight statistics to improve SQNR following ADC quantization. In this section, we study the impact of W-reshape on inference accuracy. Specifically, we use ResNet50 as an example in this study. We first visualize kurtosis with different λ_κ in Fig. 5a by computing the kurtosis for quantized weights at each layer. As seen, increasing λ_κ reduces kurtosis, but saturates when λ_κ becomes too large. We also observe that weights in the later layers are more resistive to the effects of kurtosis loss. As shown in Fig. 5b, the blue dots represent the case when we apply a constant λ_κ to all layers, where we can observe larger kurtosis in later layers. We can therefore adjust the kurtosis-loss coefficient for these layers, applying $4\times$ higher weighting, i.e.,

$$\mathcal{L}_Q = \mathcal{L}_c + \lambda_\kappa \left(\sum_{l=1}^{J} \kappa_l + 4 \sum_{l=J+1}^{L} \kappa_l \right) \tag{13}$$

where L is the number of layers, and J is the boundary to split front layers and later layers. The result is illustrated as orange dots in Fig. 5b, which show reduced kurtosis in later layers.

Table 8 and Table 9 show both the ResNet50 and MobileNetV2 accuracy of QAT (i.e., without ADC) and the accuracy after incorporating ADC quantization under different strengths of the kurtosis loss. As we can see, there is clearly a trade-off between QAT accuracy and the amount of kurtosis loss applied, which therefore impacts the overall accuracy with ADC quantization. First, we can see that small λ_κ provides slightly higher accuracy for QAT without ADC quantization. However, large kurtosis of the quantized-weight distribution leads to low variance of the IMC compute output (ADC input). Consequently, accuracy after incorporating ADC quantization is low. An extremely large λ_κ starts to degrade accuracy of QAT without ADC quantization, and thus limits the accuracy achievable after incorporating ADC quantization, despite larger variance of the IMC compute output. This can be further understood by plotting the distributions of quantized weights for each λ_κ, as shown in Fig. 5c-5e. As seen, a large λ_κ leads to significant clipping error, eliminating almost all information, and thus resulting in degraded accuracy. Therefore, in this work, we choose $\lambda_\kappa = 0.0005$ for ResNet50 and 0.00065 for MobileNetV2 to maximize the variance of quantized weights.

Table 8. ResNet accuracy with different λ_κ.

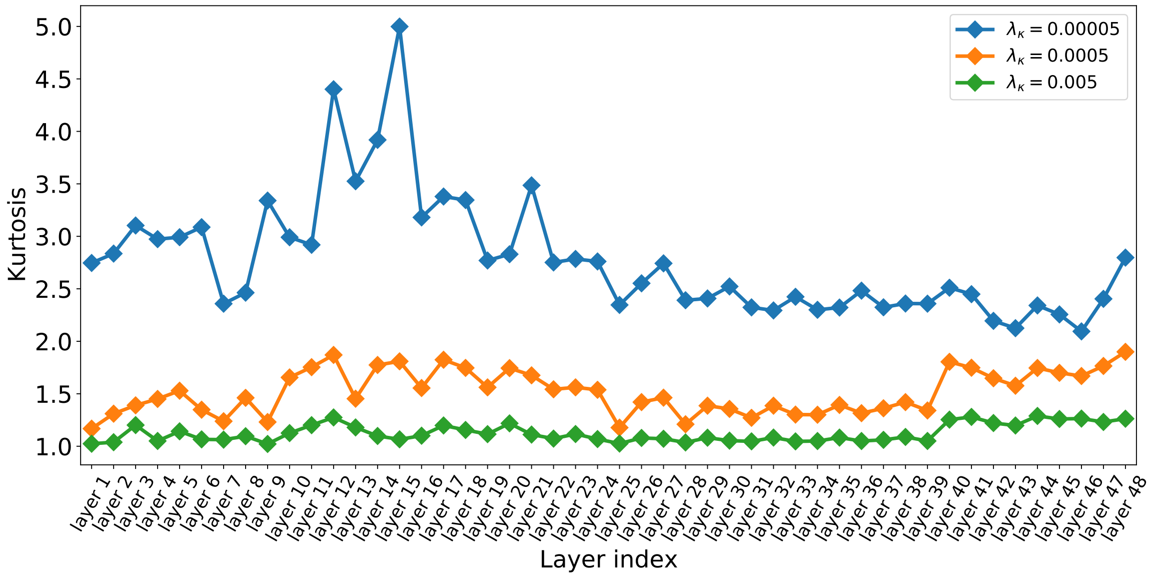
λ_κ	0	0.000025	0.00005	0.0005	0.005	0.01
ACCURACY (W/O ADC)	76.35	76.36	76.32	76.31	76.15	75.51
ACCURACY (W/ ADC)	75.91	75.92	76.05	76.27	75.77	75.02

Table 9. MobileNetv2 accuracy with different λ_κ.

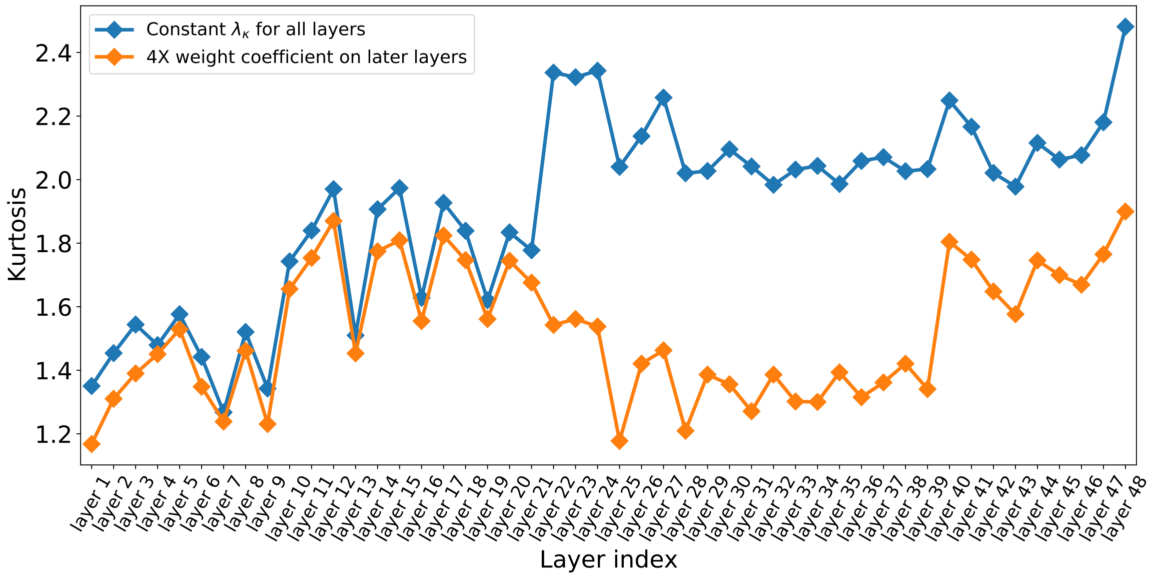
λ_κ	0	0.00004	0.00065	0.0008	0.002	0.01
ACCURACY (W/O ADC)	70.44	70.51	70.47	70.33	69.98	69.05
ACCURACY (W/ ADC)	69.92	70.02	70.46	70.24	69.65	68.86

B. IMC Compatibility

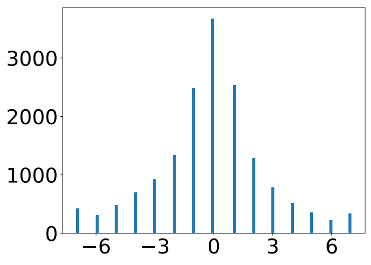
All techniques in RAOQ are compatible generally across IMC hardware. W-reshape and BitAug simply impact the weight parameters derived from neural network training. A-shift is a little different, in that it is affected by how activations are mapped for IMC computation after training, and here we examine its impact from different IMC hardware approaches. Previous IMC works employ different ways of encoding multi-bit activations and weights. For example, Dong et al. (2020) follows conventional 2's complement format, which we refer to as 0/1 representation, corresponding to the mathematical value of individual binary-weighted bits. However, other works like (Lee et al., 2021) represent a multi-bit number with individual binary-weighted bits taking mathematical values of -1 or 1, thus enabling multiplication simply by performing logical XNOR operations. We refer to this format as -1/1 representation. These two types of number representations are illustrated in Fig. 6, taking 2-bit as an example. In Section 3.2 of the paper, we show A-shift for 0/1 representation, which is the default number representation in neural network training. In fact, our proposed A-shift can be easily adapted to -1/1



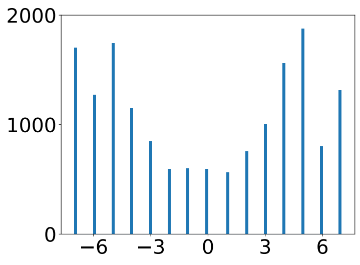
(a) Kurtosis of each layer.



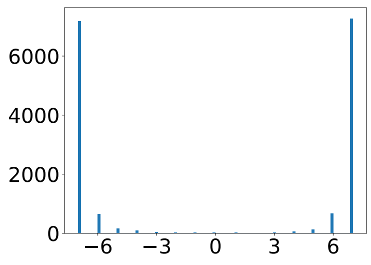
(b) Comparison of different strategies to assign λ_κ to neural network layers.



(c) $\lambda_\kappa = 0.00005$



(d) $\lambda_\kappa = 0.0005$



(e) $\lambda_\kappa = 0.005$

Figure 5. Visualize the impact of W-reshape on quantized weights.

representation as well. This is because these two representations can be converted to each other via a linear transformation. Let $\overline{x}_{0/1}$ and $\overline{x}_{-1/1}$ denote the IMC input for 0/1 representation and -1/1 representation, respectively, then:

$$\overline{x}_{-1/1} = 2\,\overline{x}_{0/1} + 1 \tag{14}$$

Therefore, A-shift for -1/1 representation can be expressed as:

$$\begin{aligned} \overline{x}_s &= 2\left\lfloor clip\left(\frac{x-\beta}{s_x}, 0, 2^{b_x}-1\right)\right\rceil - (2^{b_x}-1) \qquad (15)\\ &= 2\,\overline{x}_{-1/1} - (2^{b_x}-1) \qquad (16) \end{aligned}$$

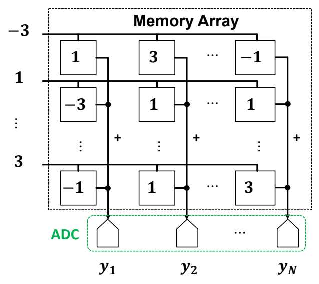
We can see that while A-shift for 0/1 representation shifts the range from $\{n : n \in \mathbb{Z} \text{ and } n \geq 0 \text{ and } n \leq 2^{bx}-1\}$ to $\{n : n \in \mathbb{Z} \text{ and } n \geq -2^{bx-1} \text{ and } n \leq 2^{bx-1}-1\}$, A-shift for -1/1 representation shifts to $\{2n+1 : n \in \mathbb{Z} \text{ and } n \geq -2^{bx-1} \text{ and } n \leq 2^{bx-1}-1\}$. Similar to the case of 0/1 representation, the extra offset introduced by A-shift can be computed offline. In summary, all of our proposed approaches are compatible with various IMC types.

Number	Representation
-2	1 0
-1	1 1
0	0 0
1	0 1

(a) 0/1 representation.

Number	Representation
-3	-1 -1
-1	-1 1
1	1 -1
3	1 1

(b) -1/1 representation.



(c) IMC.

Figure 6. (a-b) Number representations of different IMCs. (c) Example IMC system using -1/1 representation.

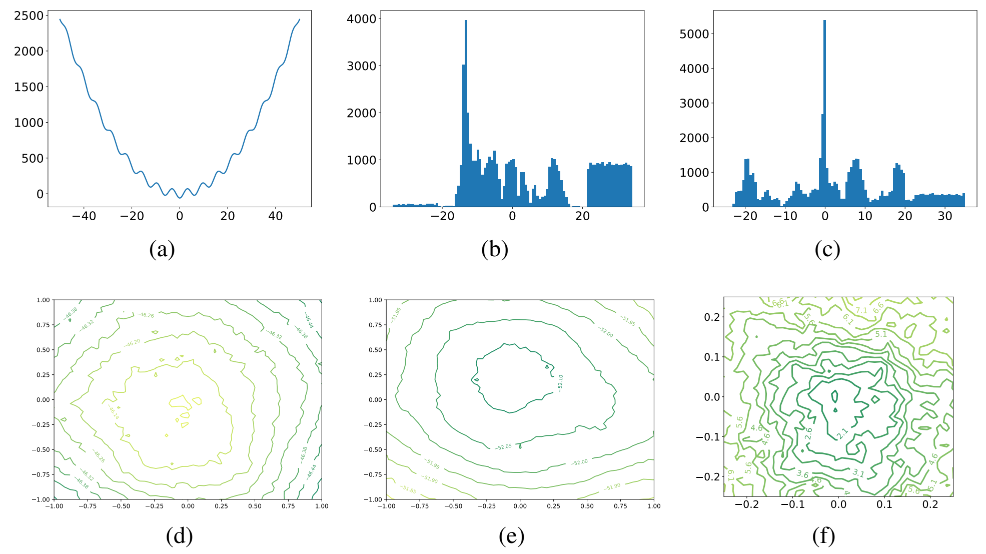


Figure 7. (a) Illustration of $g(\cdot)$. (b-c) Weight distribution collected at the last iteration without and with BitAug applied, respectively. (d-e) Loss surfaces starting from pre-trained checkpoints without and with BitAug applied, separately. (f) Loss surface of a 6-bit ADC with BitAug applied.

C. Analysis of BitAug

In this section, we provide a further study of BitAug. We argue that BitAug can assist the training process to avoid getting stuck in local minima. To demonstrate this, we consider a toy example, i.e., a single layer neural network with randomly generated input $X \in \mathbb{R}^{M\times M}$ and a randomly generated weight $W \in \mathbb{R}^{M\times N}$, followed by a simple spiking function $g(x) = x^2 - 60cos(x)$ (with global minimum at $x = 0$), as shown in Fig. 7a. We define the loss function as:

$$\mathcal{L} = \sum_{i,j} g(XW)_{i,j} \tag{17}$$

To study the effects of BitAug, we introduce quantizers for the activations, weights, and outputs of this network. Clearly, $\mathcal{L}$ gets its global minimum when all weights are zero. We first perform training without BitAug. The statistics of the weight parameters from the last iteration are shown in Fig. 7b with a standard deviation of 18.07. Rather than converging to zero, most of the weights are concentrated at some negative value, which indicates a possible local minima. Comparatively, the results after applying BitAug are shown in Fig. 7c with a standard deviation of 12.39. As seen, it shows a stronger convergence towards zero compared to without BitAug, improving the ability to escape from the local minima.

We further plot the loss surfaces at the pre-trained checkpoints for both cases (i.e., without and with BitAug), illustrated in Fig. 7d and Fig. 7e. We observe a reduction of the number local minimum after applying BitAug. Fig. 7f further confirms this observation, which shows the loss surface with 6-bit ADC and with BitAug applied. Compared to Fig. 4d where BitAug is not present, we can see a reduced number of local minimums.

D. Choosing Bit precision Candidates for BitAug

Table 10. Accuracy of different choices of candidate sets.

MODEL	$\{\}$	$\{-2,-1\}$	$\{-1,1\}$	$\{+1,+2\}$	$\{-1,1,2\}$	$\{-2,-1,+1,+2\}$	$\{-1,+1,+2,+3\}$
MOBILENETV2	68.45	66.65	68.98	68.70	69.92	69.11	69.41
RESNET50	74.47	71.12	73.37	73.42	75.83	74.77	75.21
BERT-BASE	85.45	82.13	85.41	86.88	86.92	86.04	86.58

In this section, we first explore the impact of different candidate sets. Table 10 shows the model accuracy of different candidates using MobileNetV2, ResNet50, and BERT-base as examples, with 4-bit activations/weights and 8-bit ADCs. In order to see the more obvious effects from BitAug, we use ADCs with $k = 1$, i.e., more quantization noise. Each candidate set is represented by the offset of element values from the target ADC bit precision b_a, e.g., $\{-2, -1\}$ implies $\{b_a - 2, b_a - 1\}$. As seen, $\{-1, 1, 2\}$ achieves the highest accuracy, and thus is used for our experiments in Section 4.

We also explore the relationship between the training complexity and the model accuracy. Table 11 compares the accuracy achieved by randomly sampling one element from the candidate set as well as running all elements from the candidate set simultaneously. As seen, our proposed execution of BitAug not only has lower computational complexity compared to running all bits at once, but demonstrates higher accuracy.

Table 11. Evaluation accuracy by executing BitAug in different modes.

MODE	SAMPLE SINGLE BIT	RUN ALL CANDIDATES
ACCURACY	69.92	69.04

E. Discussion of LoRA Initialization

The initialization of LoRA-based techniques is aimed at initializing its parameters A and B in a way such that the model behaves the same as before at the beginning of the finetuning, which prevents a drastic change of the model. Recent works propose to employ SVD to approximate A and B (Li et al., 2023; Guo et al., 2023) for weight quantization. However, such a method does not apply to IMC ADC quantization. The most important reason is that such an initialization only considers weight quantization and does not account for any quantization effects on the computed output (i.e., the product of activations and weights). After all, the ADC quantization is dependent on not only weight parameters but also activations.

Therefore, this work proposes an MSE-based approach to properly initialize the model for ADC quantization and minimize such quantization errors, as indicated in Eq. 12. We use the first few iterations for this regression during the RAOQ stage, and then switch back to the normal finetuning against the ground truth through the proposed methods. Our results in Section 4 have demonstrated the effectiveness of such an initialization method. Table 12 shows a comparison of our proposed ADC-LoRA with other methods, with 4-bit activations and weights, and 8-bit ADC. BERT-large is measured in F1 score, whereas OPT-1.3B and BLOOM-1.1B are evaluated in perplexity using the WikiText-2 dataset. While different approaches tend to favor different models, our proposed method consistently demonstrates high performance.

Table 12. Comparison of ADC-LoRA with other initialization methods for IMC ADC quantization.

Methods	(Hu et al., 2021)	(Li et al., 2023)	ADC-LoRA
BERT-large	89.19	89.10	89.28
OPT-1.3B	15.02	14.88	14.42
BLOOM-1.1B	21.11	20.09	20.06

F. Impact of IMC Configurations

In this section, we explore our proposed RAOQ under different hardware configurations. First, we choose ResNet50, MobileNetV2, and BERT-base for different IMC rows, corresponding to the inner-product accumulation dimension. Table 13 shows the evaluation accuracy for different memory inner-product dimensions with 4-bit activations and weights, and 8-bit ADCs having $k = 4$, demonstrating the consistently high accuracy. As we can see, our proposed RAOQ is robust across different memory sizes, indicating promise for deriving substantial benefits from in-memory computing. We can also see that different models slightly favor different memory dimensions. For example, ResNet50 and MobileNetV2 degrade slightly for the case of 256 rows, while BERT-base achieves the best accuracy in this case. This is related to the size of different neural network layers and their mapping to IMC systems, which is out of the scope of this work.

Table 13. Evaluation accuracy of different memory inner-dimensions.

# of rows	128	256	512	768	1024
ResNet50	76.33	76.28	76.27	76.31	76.24
MobileNetV2	70.53	70.40	70.46	70.45	70.43
BERT-base	87.45	87.81	87.67	87.59	87.32

Fig. 14 shows the performance of our proposed RAOQ with different values of k, i.e., the clipping set by hardware designers. As observed, these models generally favor some clipping in exchange for finer ADC quantization steps. Despite employing aggressive clipping with $k = 8$, these models preserve relatively high accuracy. This can be attributed to the fact that even with the use of our proposed W-reshape and A-shift, the distribution of ADC-input data still concentrates within a narrow portion of the entire ADC range. Therefore, ADC quantization error still dominates clipping error. However, a significant degradation in model accuracy becomes apparent when k is set to 16, even with the help of our proposed RAOQ. At this point, clipping errors start to dominant, leading to considerable loss of information.

Table 14. Evaluation accuracy of different k.

k	1	2	4	8	16
ResNet50	75.84	76.21	76.27	76.13	75.67
MobileNetV2	69.92	70.36	70.46	70.48	69.47
BERT-base	86.74	86.97	87.67	87.28	6.62

G. Comparison of QAT methods

Table 15 shows a comparison between different QAT methods based on the BERT-base model with 4-bit weights and activations, and 8-bit ADCs. The first column shows the name of each tested QAT method. The resting columns show the accuracy without ADC present, with ADC present but without RAOQ techniques, with ADC present and with RAOQ applied, respectively. As seen, RAOQ demonstrates a stable behavior on all of these methods, significantly restoring their performance to baseline level.

Table 15. Comparison of different QAT methods.

METHODS	QAT WITHOUT ADC	QAT ONLY	RAOQ
LSQ+ (BHALGAT ET AL., 2020)	87.75	82.43	87.67
LSQ (ESSER ET AL., 2019)	87.60	82.02	87.41
PACT+SAWB (CHOI ET AL., 2019)	87.49	80.88	87.34

H. Comparison with IMC QAT Methods on More Complex Tasks

Although prior IMC QAT works only demonstrated the results on simple tasks and datasets as mentioned in Section 4.2, e.g., image classification on CIFAR-10, we implemented these methods by ourselves and made a comparison between these works and our proposed RAOQ approach to complete our study. This is illustrated in Table 16-17. ResNet18/50, MobileNetv2, and EfficientNet-lite0 are evaluated on ImageNet (accuracy); YOLOv5s is evaluated on COCO 2017 (mAP); BERT family is evaluated on SQuAD 1.1 (F1 score). Higher values for these metrics indicate better results. On the other hand, OPT and BLOOM models are evaluated on WikiText-2 (perplexity), where lower values indicate better results. As seen, our proposed approach consistently outperforms in all cases.

Table 16. Comparison of different IMC QAT methods with 4-bit weights and activations, and 8-bit ADCs on CNN models.

	RESNET18/50	MOBILENETV2	EFFICIENTNET-LITE0	YOLOV5S
(JIN ET AL., 2022)	68.98 / 75.14	67.67	68.54	30.24
(SUN ET AL., 2021)	69.24 / 75.58	66.02	67.46	31.84
(WEI ET AL., 2020)	66.64 / 74.32	10.06	14.75	30.12
RAOQ	70.45 / 76.27	70.46	72.76	33.49

Table 17. Comparison of different IMC QAT methods with 4-bit weights and activations, and 8-bit ADCs on transformer models.

	BERT-BASE/LARGE	OPT-125M/350M/1.3B	BLOOM-560M/1.1B/1.7B
(JIN ET AL., 2022)	86.23 / 87.54	38.12 / 29.26 / 21.34	42.12 / 33.12 / 28.14
(SUN ET AL., 2021)	87.02 / 88.52	30.13 / 23.56 / 19.76	32.71 / 26.45 / 21.02
(WEI ET AL., 2020)	22.12 / 22.34	476.68 / 250.12 / 98.86	122.02 / 211.64 / 102.64
RAOQ	87.67 / 89.28	24.71 / 19.88 / 14.42	24.94 / 20.06 / 17.06

I. Framework

Algorithm 1 summarizes our training framework for IMC ADC quantization, including the proposed A-shift, W-reshape, BitAug, and ADC-LoRA approaches. All implementations are done in PyTorch. Once the model is trained, it is employed for inference using IMC, as shown in Fig. 8. For the IMC inference, the input data and model parameters are first transferred to the IMC system. The output is then collected and sent back to the host for further processing.

J. Training Details

In this section, we describe the models, datasets, and hyperparameter settings used in our experiments. We implement our models in the PyTorch framework. The first and last layers are kept in 8-bit, and are not mapped to the IMC. Mapping these layers to IMC provides marginal benefit, since the first layers have few input channels and thus limited opportunity for row parallelism, while the last layers have low data reuse, contributing a small number of total operations and restricting amortization of the IMC weight-loading overheads. Also, to preserve the fidelity of critical information, we do not map depthwise convolutions in the MobileNet family, and the second matrix-matrix multiplication in the self-attention module of transformer models (BMM2) to the IMC system. This is justified as these layers account for a small number of computations

Algorithm 1 RAOQ for IMC ADC Quantization. J is the number of layers, $Q_X(\cdot)$ and $Q_W(\cdot)$ are conventional quantizers for activations and weights, respectively, $Q_A(\cdot)$ is the ADC quantizer defined in Eq. 2, s_w, s_x, and β are quantization parameters, τ denotes the initialization duration for ADC-LoRA, I_Q and I_A denote the total number of iterations for QAT and ADC phases, separately.

```
Input: pre-trained floating-point model, input $x$
{QAT stage}
for $i = 1$ to $I_Q$ do
  $\mathcal{L}_\kappa \leftarrow 0$
  for $j = 1$ to $J$ do
    $\overline{x} \leftarrow Q_X(x, b_x, s_x, \beta)$
    $\overline{w} \leftarrow Q_W(w, b_w, s_w)$
    $y \leftarrow \text{MVM}(\overline{x}, \overline{w})$
    $\mathcal{L}_\kappa \leftarrow \mathcal{L}_\kappa + \kappa(\overline{w})$        # $\kappa$ is defined in Eq. 6 for W-reshape
  end for
  Compute the loss $\mathcal{L}_c$ against the ground truth
  $\mathcal{L}_Q \leftarrow \mathcal{L}_c + \lambda_\kappa \mathcal{L}_\kappa$
  Backprop based on $\mathcal{L}_Q$ and update model parameters
end for
Collect updated model parameters
{RAOQ stage}
for $i = 1$ to $\tau$ do
  Initialize LoRA parameters $A$ and $B$ via ADC-LoRA defined in Eq. 12
end for
for $i = \tau + 1$ to $I_A$ do
  for $j = 1$ to $J$ do
    $\overline{x} \leftarrow Q_X(x, b_x, s_x, \beta) - 2^{b_x - 1}$        # for A-shift
    $\overline{w} \leftarrow Q_W(w, A, B, b_w, s_w)$
    $y \leftarrow \text{IMC}(\overline{x}, \overline{w})$     #  IMC($\cdot$) indicates performing computation in IMC systems
    $\overline{y} \leftarrow Q_A(y, b_a)$
  end for
  Compute the loss $\mathcal{L}_c(b_a)$ against the ground truth based on $b_a$ and backprop
  Sample $\widetilde{b}_a$ from candidate set $\mathbb{B}$       # prepare for BitAug
  for $j = 1$ to $J$ do
    $\overline{x} \leftarrow Q_X(x, b_x, s_x, \beta) - 2^{b_x - 1}$
    $\overline{w} \leftarrow Q_W(w, A, B, b_w, s_w)$
    $y \leftarrow \text{IMC}(\overline{x}, \overline{w})$     #  IMC($\cdot$) indicates performing computation in IMC systems
    $\overline{y} \leftarrow Q_A(y, \widetilde{b}_a)$
  end for
  Compute the loss $\mathcal{L}_c(\widetilde{b}_a)$ against the ground truth based on $\widetilde{b}_a$ and backprop
  $\mathcal{L}_A \leftarrow \mathcal{L}_c(b_a) + \lambda_b \mathcal{L}_c(\widetilde{b}_a)$
  Accumulate gradients from BitAug and update model parameters (Eq. 8)
end for
```

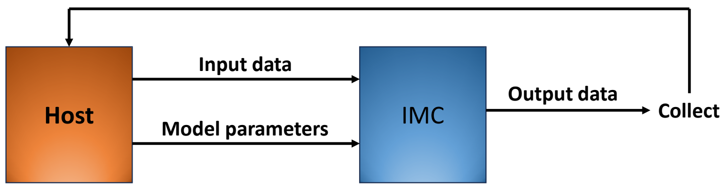


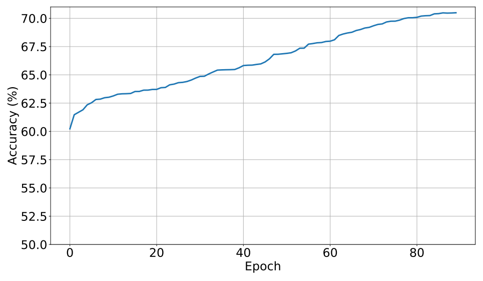
Figure 8. Inference flow using IMC.

in the overall model (e.g., $< 7\%$ for depthwise convolutions in MobileNetV2 and $< 1.5\%$ for BMM2 in BERT), thus giving minor energy-efficiency advantage by execution via IMC. λ_b for BitAug is initialized to be 1, and drops following a cosine scheduling for all models. We first perform QAT for these models based on LSQ+ (Bhalgat et al., 2020) without ADC quantization. Specifically, we perform per-tensor quantization for both activations and weights for CNN models and encoder models. For decoder models (OPT and BLOOM), we apply per-tensor quantization for activations and per-channel quantization for weights due to their high sensitivity to quantization. Then we perform our proposed RAOQ with ADC quantization incorporated. In the following sections, we show the training curves for both QAT stage (without ADC quantization) and the training stage with ADC quantization involved for each model, with 4-bit activations and weights, and 8-bit ADCs as examples.

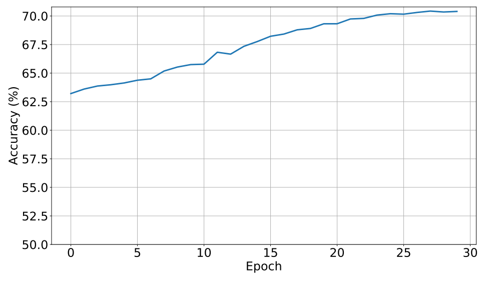
J.1. Image Classification

We perform image classification using the ImageNet dataset (Deng et al., 2009). Our experiments consist of models from the ResNet family and the MobileNet family, whose training settings are discussed in detail as follows.

ResNet18. For 4-bit QAT (i.e., 4-bit activations and weights), we perform training for 90 epochs, with a batch size of 256. We use SGD optimizer with a momentum of 0.9 and weight decay of 0.0001. The learning rate starts at 0.01 and gradually drops, following a cosine annealing scheduler. λ_κ is set to 0.002. For 8-bit QAT, we follow the same optimizer and batch size as the 4-bit case. We train for 30 epochs with an initial learning rate of 0.005. λ_κ is set to 0.0014. When ADC quantization is added to the model, we perform another 30-epoch training, using the cosine annealing learning rate scheduler with an initial learning rate of 0.004. The optimizer and batch size remain the same as those used in the previous QAT stage. All experiments for ResNet18 are conducted on 2 Nvidia A100 GPUs.



(a) QAT without ADC



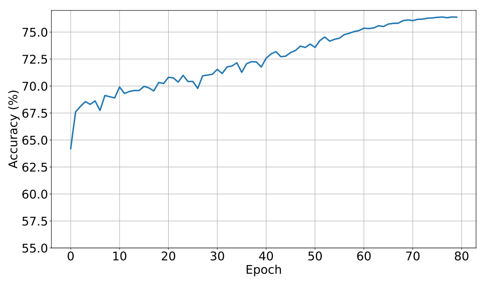
(b) RAOQ with ADC

Figure 9. Training curves for ResNet18.

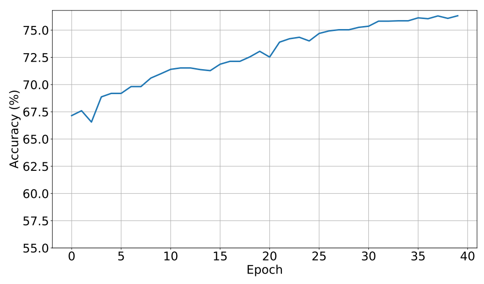
ResNet50. During the QAT stage, we train for 80 epochs, with batch size 256 for 4-bit activations and weights. We use the same optimizer and learning rate scheduler as used for ResNet18. λ_κ for ResNet50 is set to 0.0005. For 8-bit QAT, we maintain the same optimizer and learning rate scheduler, but with a different initial learning rate of 0.002. We train for 40 epochs with a batch size of 256. λ_κ is set to 0.00011. For the next stage incorporating ADC quantization, we train for another 40 epochs with an initial learning rate of 0.002. The rest of the settings are the same as those used for ResNet18. All experiments for ResNet50 are performed on 2 Nvidia A100 GPUs.

MobileNetV2. We perform 4-bit QAT for 70 epochs with batch size 256. We use SGD optimizer with a momentum of 0.9 and weight decay of 0.00004. The initial learning rate is set to 0.01, with a cosine annealing scheduler. λ_κ is set to 0.00065. For 8-bit QAT, we keep the same optimizer, scheduler, λ_κ, and batch size, training for 40 epochs with an initial learning rate of 0.002. We then perform another stage of training with ADC quantization added for 50 epochs. We use the same optimizer, scheduler, and batch size as used in QAT. The learning rate starts at 0.004. The entire training for MobileNetV2 is on 2 Nvidia A100 GPUs.

EfficientNet-lite0. The settings are the same as MobileNetV2, except that λ_κ is set to 0.002. We perform training for 80 epochs with an initial learning rate of 0.01 for both 4-bit and 8-bit QAT. When ADC quantization is added for the subsequent training phase, we use the same optimizer, scheduler, and batch size, running for 50 epochs. The initial learning rate is set to 0.004. Once again, we perform the experiments on 2 Nvidia A100 GPUs.

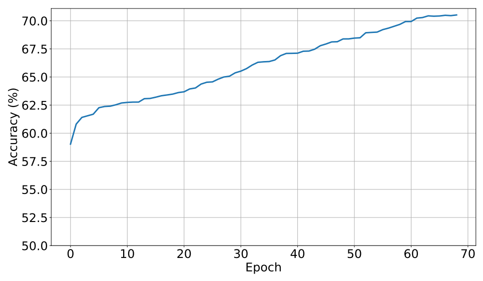


(a) QAT without ADC

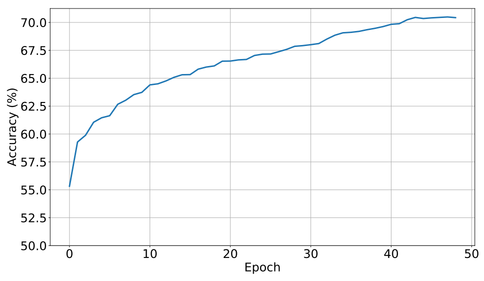


(b) RAOQ with ADC

Figure 10. Training curves for ResNet50.

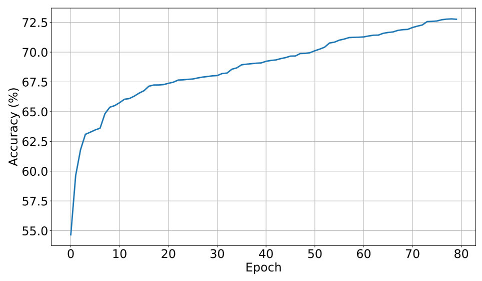


(a) QAT without ADC

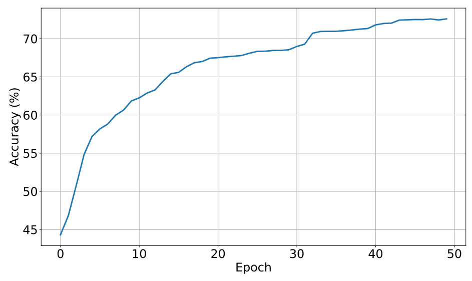


(b) RAOQ with ADC

Figure 11. Training curves for MobileNetV2.



(a) QAT without ADC



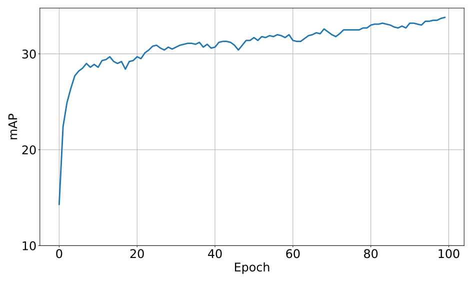
(b) RAOQ with ADC

Figure 12. Training curves for EfficientNet-lite0.

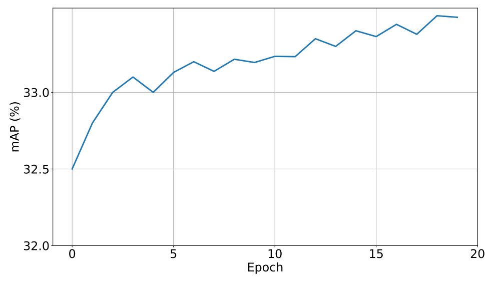
J.2. Object Detection

We perform object detection on the COCO 2017 dataset (Lin et al., 2014). YOLOv5s is used to conduct the task.

YOLOv5s. We first retrain the floating point (FP) model by using hyperparameters indicated in (Jocher et al., 2022). However, we remove the automatic mixed precision (AMP) features supported by PyTorch, since we would like to perform quantization to the target bit precisions later on. Most of our hyperparameters remain the same as (Jocher et al., 2022) suggested, and we specify those we customized for this work here. For 4-bit QAT, we train for 100 epochs with a batch size of 64 with a weight decay of 0.0001. The initial learning rate is set to 0.004 following a cosine scheduler, and the momentum for the optimizer is changed to 0.9 from 0.937 in (Jocher et al., 2022). The 8-bit QAT runs for 80 epochs with the same optimizer, scheduler, and batch size as used for the 4-bit QAT, and an adjusted initial learning rate of 0.004. λ_κ is set to 0.0001. For training with ADC quantization incorporated, we perform the training for 20 epochs for the 4-bit case, with the initial learning rate of 0.0001 and weight decay of 0.00005. We train the 8-bit YOLO model with ADC quantization for 90 epochs, using the same hyperparameters as used in the QAT stage except for the batch size increased to 128. Our experiments are performed on 4 Nvidia A100 GPUs.



(a) QAT without ADC



(b) RAOQ with ADC

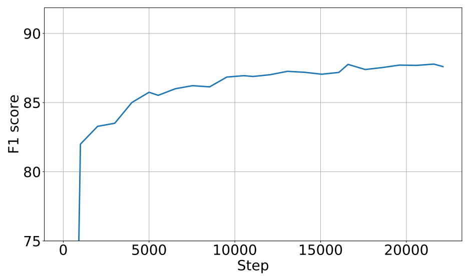
Figure 13. Training curves for YOLOv5s.

J.3. Natural Language Processing (NLP)

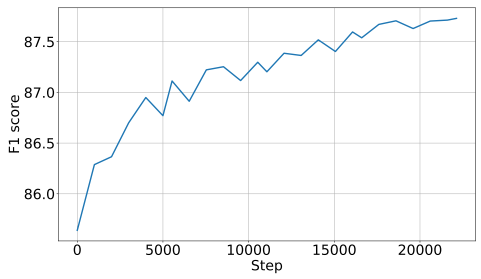
We perform the question-answering task on SQuAD 1.1 (Rajpurkar et al., 2016) for encoder models (BERT), and on the language modeling task on WikiText-2 and WikiText-103 (Merity et al., 2016) for decoder models (OPT and BLOOM). As only a small number of activations in the these models follow non-linear functions (e.g., GELU), we can apply A-shift to only these layers. Activations from other layers are directly taken as the result of matrix multiplications. We find that forcing their quantization to unsigned numbers causes accuracy degradation. For encoder models, we first finetune the pre-trained FP models to the downstream SQuAD 1.1 dataset before quantization gets involved, as suggested by (Wang et al., 2022). For decoder models, we skip the finetuning stage for FP models, directly quantizing the pre-trained model to the downstream task. We use AdamW optimizer for the following experiments.

BERT-base/large. We use a batch size of 16, running for 4 epochs. The initial learning rate is kept at 0.00003, following a linear decay. The dropout rate is raised to 0.2 for BERT-base. We observe that 4-bit QAT for BERT-large is sensitive to the change in dropout rate. Thus, we start with a dropout rate of 0.1 for the first epoch and then raise to 0.2 for the rest of the training. The RAOQ stage employs the same hyperparameters used in the QAT stage. Experiments are performed on 2 Nvidia A100 GPUs.

OPT-125m/350m/1.3B. During the QAT stage, we use a batch size of 8, 2, 2 for OPT-125m, OPT-350m, and OPT-1.3B, respectively. To maintain reasonable memory usage, we apply LoRA for OPT-350m and OPT-1.3B, and initialize LoRA parameters as a Gaussian and a zero tensor as Hu et al. (2021) suggested, with a rank of 64. The initial learning rate is set to 0.00006 for OPT-125m and 0.0002 for the other two OPT variants. We train 6 epochs for WikiText-2 and 1 epoch for WikiText-103. Additionally, we apply an extra dropout rate of 0.1 to the attention modules to prevent overfitting. In terms of the RAOQ stage, hyperparameters maintain the same, tuning for another 3 epochs for WikiText-2 and 1 epoch for WikiText-103. New LoRA parameters are introduced and tuned via the proposed ADC-LoRA with the rank of 32. Experiments are performed on 2 Nvidia A100 GPUs.

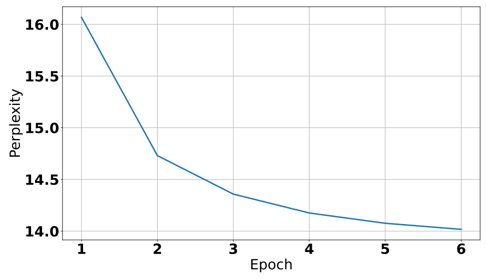


(a) QAT without ADC

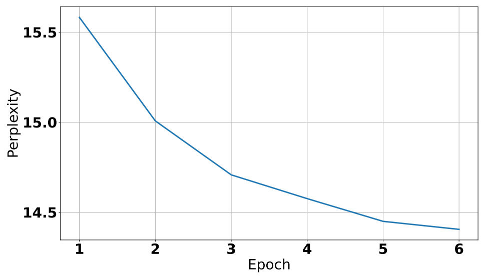


(b) RAOQ with ADC

Figure 14. Training curves for BERT-base.



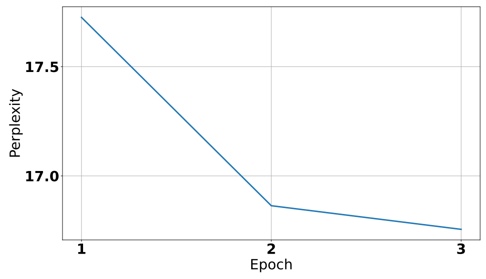
(a) QAT without ADC



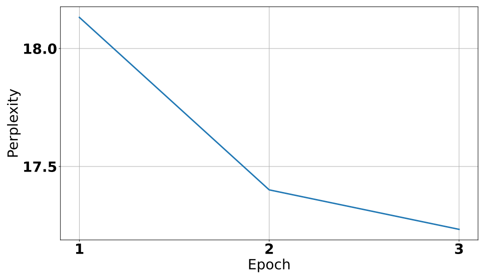
(b) RAOQ with ADC

Figure 15. Training curves for OPT-1.3B.

BLOOM-560m/1.1B/1.7B. We use a batch size of 2 for all BLOOM models and all training stages. During the QAT stage, we apply LoRA with a rank of 64. The initial learning rate is set to 0.0001, training for 3 epochs for WikiText-2 and 1 epoch for WikiText-103. The RAOQ stage uses the same hyperparameters as the QAT stage, training for another 3 epochs for WikiText-2 and 1 epoch for WikiText-103. We set the rank of LoRA parameters to 32 for this stage, initialized using our proposed ADC-LoRA. Experiments are performed on a single Nvidia A100 GPU.



(a) QAT without ADC

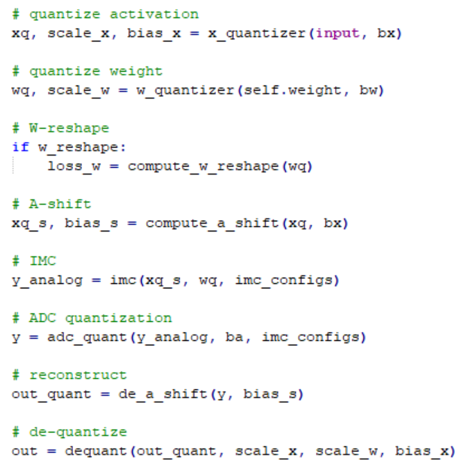


(b) RAOQ with ADC

Figure 16. Training curves for BLOOM-1.7B.

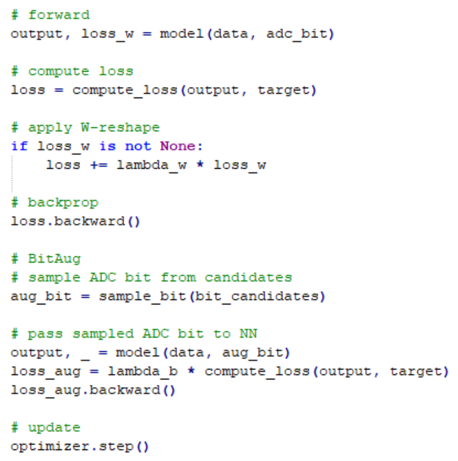
J.4. Code example

All models are implemented using the PyToch framework. Specifically, we customized nn.Conv2d and nn.Linear modules to incorporate quantization of activations/weights/ADCs, mapping to IMC systems, and our proposed A-shift and W-reshape methods. The proposed BitAug technique is integrated with top-level training. Fig. 17a-17b shows example code snippets for a convolution layer with all quantization sources integrated and the implementation of our proposed approaches. Fig. 17c illustrates a screenshot of the training log of MobileNetV2.



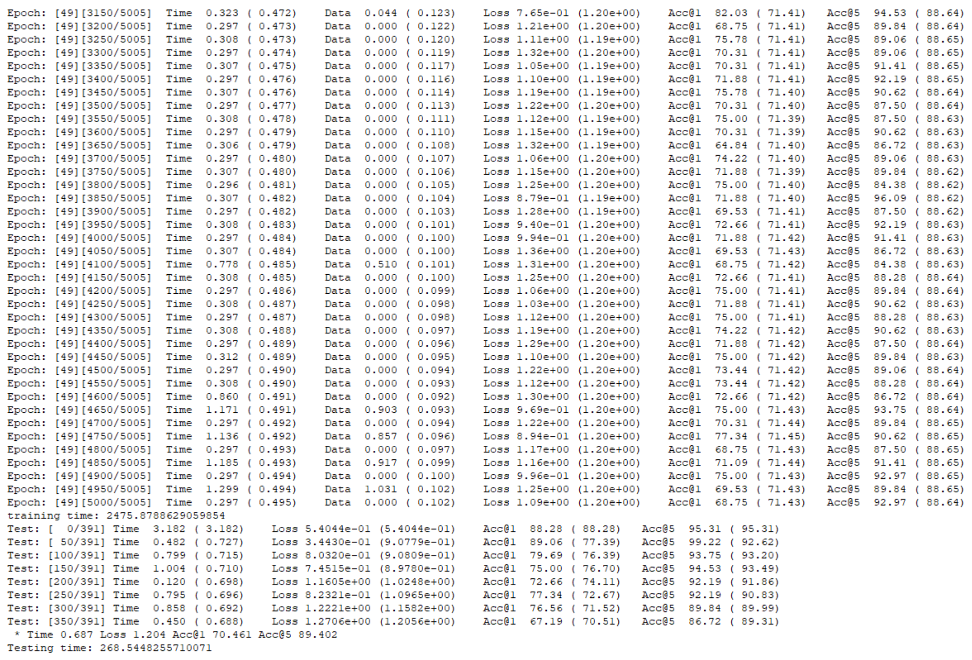
```
# quantize activation
xq, scale_x, bias_x = x_quantizer(input, bx)

# quantize weight
wq, scale_w = w_quantizer(self.weight, bw)

# W-reshape
if w_reshape:
    loss_w = compute_w_reshape(wq)

# A-shift
xq_s, bias_s = compute_a_shift(xq, bx)

# IMC
y_analog = imc(xq_s, wq, imc_configs)

# ADC quantization
y = adc_quant(y_analog, ba, imc_configs)

# reconstruct
out_quant = de_a_shift(y, bias_s)

# de-quantize
out = dequant(out_quant, scale_x, scale_w, bias_x)
```

(a)



```
# forward
output, loss_w = model(data, adc_bit)

# compute loss
loss = compute_loss(output, target)

# apply W-reshape
if loss_w is not None:
    loss += lambda_w * loss_w

# backprop
loss.backward()

# BitAug
# sample ADC bit from candidates
aug_bit = sample_bit(bit_candidades)

# pass sampled ADC bit to NN
output, _ = model(data, aug_bit)
loss_aug = lambda_b * compute_loss(output, target)
loss_aug.backward()

# update
optimizer.step()
```

(b)



```
Epoch: [49][3150/5005]  Time  0.323 ( 0.472)    Data  0.044 ( 0.123)    Loss 7.65e-01 (1.20e+00)    Acc@1  82.03 ( 71.41)   Acc@5  94.53 ( 88.64)
Epoch: [49][3200/5005]  Time  0.297 ( 0.473)    Data  0.000 ( 0.122)    Loss 1.21e+00 (1.20e+00)    Acc@1  68.75 ( 71.41)   Acc@5  89.84 ( 88.64)
Epoch: [49][3250/5005]  Time  0.308 ( 0.473)    Data  0.000 ( 0.120)    Loss 1.11e+00 (1.19e+00)    Acc@1  75.78 ( 71.41)   Acc@5  89.06 ( 88.65)
Epoch: [49][3300/5005]  Time  0.297 ( 0.474)    Data  0.000 ( 0.119)    Loss 1.32e+00 (1.20e+00)    Acc@1  70.31 ( 71.41)   Acc@5  89.06 ( 88.65)
Epoch: [49][3350/5005]  Time  0.307 ( 0.475)    Data  0.000 ( 0.117)    Loss 1.05e+00 (1.19e+00)    Acc@1  70.31 ( 71.41)   Acc@5  91.41 ( 88.65)
Epoch: [49][3400/5005]  Time  0.297 ( 0.476)    Data  0.000 ( 0.116)    Loss 1.10e+00 (1.19e+00)    Acc@1  71.88 ( 71.41)   Acc@5  92.19 ( 88.65)
Epoch: [49][3450/5005]  Time  0.307 ( 0.476)    Data  0.000 ( 0.114)    Loss 1.19e+00 (1.19e+00)    Acc@1  75.78 ( 71.40)   Acc@5  90.62 ( 88.64)
Epoch: [49][3500/5005]  Time  0.297 ( 0.477)    Data  0.000 ( 0.113)    Loss 1.22e+00 (1.20e+00)    Acc@1  70.31 ( 71.40)   Acc@5  87.50 ( 88.64)
Epoch: [49][3550/5005]  Time  0.308 ( 0.478)    Data  0.000 ( 0.111)    Loss 1.12e+00 (1.19e+00)    Acc@1  75.00 ( 71.39)   Acc@5  87.50 ( 88.63)
Epoch: [49][3600/5005]  Time  0.297 ( 0.479)    Data  0.000 ( 0.110)    Loss 1.15e+00 (1.19e+00)    Acc@1  70.31 ( 71.39)   Acc@5  90.62 ( 88.63)
Epoch: [49][3650/5005]  Time  0.306 ( 0.479)    Data  0.000 ( 0.108)    Loss 1.32e+00 (1.19e+00)    Acc@1  64.84 ( 71.40)   Acc@5  86.72 ( 88.63)
Epoch: [49][3700/5005]  Time  0.297 ( 0.480)    Data  0.000 ( 0.107)    Loss 1.06e+00 (1.20e+00)    Acc@1  74.22 ( 71.40)   Acc@5  89.06 ( 88.63)
Epoch: [49][3750/5005]  Time  0.307 ( 0.480)    Data  0.000 ( 0.106)    Loss 1.15e+00 (1.20e+00)    Acc@1  71.88 ( 71.39)   Acc@5  89.84 ( 88.62)
Epoch: [49][3800/5005]  Time  0.296 ( 0.481)    Data  0.000 ( 0.105)    Loss 1.25e+00 (1.20e+00)    Acc@1  75.00 ( 71.40)   Acc@5  84.38 ( 88.62)
Epoch: [49][3850/5005]  Time  0.307 ( 0.482)    Data  0.000 ( 0.104)    Loss 8.79e-01 (1.19e+00)    Acc@1  71.88 ( 71.40)   Acc@5  96.09 ( 88.62)
Epoch: [49][3900/5005]  Time  0.297 ( 0.482)    Data  0.000 ( 0.103)    Loss 1.28e+00 (1.19e+00)    Acc@1  69.53 ( 71.41)   Acc@5  87.50 ( 88.62)
Epoch: [49][3950/5005]  Time  0.308 ( 0.483)    Data  0.000 ( 0.101)    Loss 9.40e-01 (1.20e+00)    Acc@1  72.66 ( 71.41)   Acc@5  92.19 ( 88.63)
Epoch: [49][4000/5005]  Time  0.297 ( 0.484)    Data  0.000 ( 0.100)    Loss 9.94e-01 (1.20e+00)    Acc@1  71.88 ( 71.42)   Acc@5  91.41 ( 88.63)
Epoch: [49][4050/5005]  Time  0.307 ( 0.484)    Data  0.000 ( 0.100)    Loss 1.36e+00 (1.20e+00)    Acc@1  69.53 ( 71.43)   Acc@5  86.72 ( 88.63)
Epoch: [49][4100/5005]  Time  0.778 ( 0.485)    Data  0.510 ( 0.101)    Loss 1.31e+00 (1.20e+00)    Acc@1  68.75 ( 71.42)   Acc@5  84.38 ( 88.63)
Epoch: [49][4150/5005]  Time  0.308 ( 0.485)    Data  0.000 ( 0.100)    Loss 1.25e+00 (1.20e+00)    Acc@1  72.66 ( 71.41)   Acc@5  88.28 ( 88.64)
Epoch: [49][4200/5005]  Time  0.297 ( 0.486)    Data  0.000 ( 0.099)    Loss 1.06e+00 (1.20e+00)    Acc@1  75.00 ( 71.41)   Acc@5  89.84 ( 88.64)
Epoch: [49][4250/5005]  Time  0.308 ( 0.487)    Data  0.000 ( 0.098)    Loss 1.03e+00 (1.20e+00)    Acc@1  71.88 ( 71.41)   Acc@5  90.62 ( 88.63)
Epoch: [49][4300/5005]  Time  0.297 ( 0.487)    Data  0.000 ( 0.098)    Loss 1.12e+00 (1.20e+00)    Acc@1  75.00 ( 71.41)   Acc@5  88.28 ( 88.63)
Epoch: [49][4350/5005]  Time  0.308 ( 0.488)    Data  0.000 ( 0.097)    Loss 1.19e+00 (1.20e+00)    Acc@1  74.22 ( 71.42)   Acc@5  90.62 ( 88.63)
Epoch: [49][4400/5005]  Time  0.297 ( 0.489)    Data  0.000 ( 0.096)    Loss 1.29e+00 (1.20e+00)    Acc@1  71.88 ( 71.42)   Acc@5  87.50 ( 88.64)
Epoch: [49][4450/5005]  Time  0.312 ( 0.489)    Data  0.000 ( 0.095)    Loss 1.10e+00 (1.20e+00)    Acc@1  75.00 ( 71.42)   Acc@5  89.84 ( 88.63)
Epoch: [49][4500/5005]  Time  0.297 ( 0.490)    Data  0.000 ( 0.094)    Loss 1.22e+00 (1.20e+00)    Acc@1  73.44 ( 71.42)   Acc@5  89.06 ( 88.64)
Epoch: [49][4550/5005]  Time  0.308 ( 0.490)    Data  0.000 ( 0.093)    Loss 1.12e+00 (1.20e+00)    Acc@1  73.44 ( 71.42)   Acc@5  88.28 ( 88.64)
Epoch: [49][4600/5005]  Time  0.860 ( 0.491)    Data  0.000 ( 0.092)    Loss 1.30e+00 (1.20e+00)    Acc@1  72.66 ( 71.42)   Acc@5  86.72 ( 88.64)
Epoch: [49][4650/5005]  Time  1.171 ( 0.491)    Data  0.903 ( 0.093)    Loss 9.69e-01 (1.20e+00)    Acc@1  75.00 ( 71.43)   Acc@5  93.75 ( 88.64)
Epoch: [49][4700/5005]  Time  0.297 ( 0.492)    Data  0.000 ( 0.094)    Loss 1.22e+00 (1.20e+00)    Acc@1  70.31 ( 71.44)   Acc@5  89.84 ( 88.65)
Epoch: [49][4750/5005]  Time  1.136 ( 0.492)    Data  0.857 ( 0.096)    Loss 8.94e-01 (1.20e+00)    Acc@1  77.34 ( 71.45)   Acc@5  90.62 ( 88.65)
Epoch: [49][4800/5005]  Time  0.297 ( 0.493)    Data  0.000 ( 0.097)    Loss 1.17e+00 (1.20e+00)    Acc@1  68.75 ( 71.43)   Acc@5  87.50 ( 88.65)
Epoch: [49][4850/5005]  Time  1.185 ( 0.493)    Data  0.917 ( 0.099)    Loss 1.16e+00 (1.20e+00)    Acc@1  71.09 ( 71.44)   Acc@5  91.41 ( 88.65)
Epoch: [49][4900/5005]  Time  0.297 ( 0.494)    Data  0.000 ( 0.100)    Loss 9.96e-01 (1.20e+00)    Acc@1  75.00 ( 71.43)   Acc@5  92.97 ( 88.65)
Epoch: [49][4950/5005]  Time  1.299 ( 0.494)    Data  1.031 ( 0.102)    Loss 1.25e+00 (1.20e+00)    Acc@1  69.53 ( 71.43)   Acc@5  89.84 ( 88.65)
Epoch: [49][5000/5005]  Time  0.297 ( 0.495)    Data  0.000 ( 0.102)    Loss 1.09e+00 (1.20e+00)    Acc@1  68.75 ( 71.43)   Acc@5  92.97 ( 88.64)
training time: 2475.8788629059854
Test: [  0/391] Time  3.182 ( 3.182)    Loss 5.4044e-01 (5.4044e-01)    Acc@1  88.28 ( 88.28)   Acc@5  95.31 ( 95.31)
Test: [ 50/391] Time  0.482 ( 0.727)    Loss 3.4430e-01 (9.0779e-01)    Acc@1  89.06 ( 77.39)   Acc@5  99.22 ( 92.62)
Test: [100/391] Time  0.799 ( 0.715)    Loss 8.0320e-01 (9.0809e-01)    Acc@1  79.69 ( 76.39)   Acc@5  93.75 ( 93.20)
Test: [150/391] Time  1.004 ( 0.710)    Loss 7.4515e-01 (8.9780e-01)    Acc@1  75.00 ( 76.70)   Acc@5  94.53 ( 93.49)
Test: [200/391] Time  0.120 ( 0.698)    Loss 1.1605e+00 (1.0248e+00)    Acc@1  72.66 ( 74.11)   Acc@5  92.19 ( 91.86)
Test: [250/391] Time  0.795 ( 0.696)    Loss 8.2321e-01 (1.0965e+00)    Acc@1  77.34 ( 72.67)   Acc@5  92.19 ( 90.83)
Test: [300/391] Time  0.858 ( 0.692)    Loss 1.2221e+00 (1.1582e+00)    Acc@1  76.56 ( 71.52)   Acc@5  89.84 ( 89.99)
Test: [350/391] Time  0.450 ( 0.688)    Loss 1.2706e+00 (1.2056e+00)    Acc@1  67.19 ( 70.51)   Acc@5  86.72 ( 89.31)
 * Time 0.687 Loss 1.204 Acc@1 70.461 Acc@5 89.402
Testing time: 268.5448255710071
```

(c)

Figure 17. (a) Code for Conv2d module with all sources of quantization, proposed W-reshape and A-shift implemented. (b) Code for top-level training with the proposed BitAug implemented. (c) Example training log for MobileNetV2.