PIM-QAT: NEURAL NETWORK QUANTIZATION FOR PROCESSING-IN-MEMORY (PIM) SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Processing-in-memory (PIM), an increasingly studied neuromorphic hardware, promises orders of energy and throughput improvements for deep learning inference. Leveraging the massively parallel and efficient analog computing inside memories, PIM circumvents the bottlenecks of data movements in conventional digital hardware. However, an **extra quantization** step (i.e. PIM quantization), typically with limited resolution due to hardware constraints, is required to convert the analog computing results into digital domain. Meanwhile, non-ideal effects extensively exist in PIM quantization because of the imperfect analog-to-digital interface, which further compromises the inference accuracy. Due to hardware limitations, PIM systems decompose the bulky matrix multiplication into smaller subsets, making the computing flow fundamentally different from the conventionally quantized models. In this paper, we propose a method for training quantized networks to incorporate PIM quantization, which is ubiquitous to all PIM systems. Specifically, we propose a PIM quantization aware training (PIM-QAT) algorithm, and introduce rescaling techniques during backward and forward propagation by analyzing the training dynamics to facilitate training convergence. We also propose two techniques, namely batch normalization (BN) calibration and adjusted precision training, to suppress the adverse effects of non-ideal linearity and stochastic thermal noise involved in real PIM chips. Our method is validated on three mainstream PIM decomposition schemes, and **physically on a prototype chip**. Comparing with directly deploying conventionally trained quantized model on PIM systems, which does not take into account this extra quantization step and thus fails, our method provides significant improvement. It also achieves comparable inference accuracy on PIM systems as that of conventionally quantized models on digital hardware, across CIFAR10 and CIFAR100 datasets using various network depths for the most popular network topology.

1 INTRODUCTION

Recent progress of deep learning has witnessed great success in a wide range of applications at the cost of enormous computations and energy budget. To alleviate the resource constraints and enable deep learning inference on pervasive mobile and edge devices, extensive research has been conducted on algorithm optimizations for conventional digital hardware (e.g. GPU, CPU), with the goals of compressing models and reducing the number of operations (Choi et al., 2018; Jin et al., 2019; 2020; Liu et al., 2020; Ye et al., 2018; Zhang et al., 2018; Zhou et al., 2016). On the other hand, hardware innovations for deep learning focus on building dedicated devices with optimized dataflow and reusing to minimize data movement (Chen et al., 2016; 2014; Du et al., 2015; Jouppi et al., 2017), which is the well known energy and latency bottleneck in deep learning and many other data-centric computations (Horowitz, 2014).

Processing in-memory (PIM), inspired by neuromorphic engineering, attracts increasing attention as a potential hardware solution to data movement bottlenecks (Ambrogio et al., 2018; Ielmini and Wong, 2018; Jia et al., 2020; Prezioso et al., 2015; Xue et al., 2020; Yao et al., 2020; Zhang et al., 2017). By performing computations directly inside the weight storage memories, PIM promises significantly reduced data traffic between the memory and computing units. The merits of PIM over conventional digital hardware are three folds. First, the data movement energy and latency can be alleviated. Second, massively parallel computing, like multiply-and-accumulate (MAC), in memory arrays greatly amortize total energy and area. Third, the computation in memory is essentially in

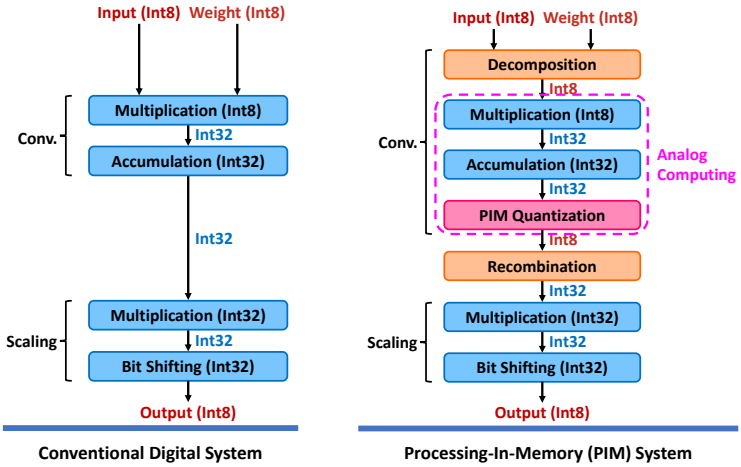

Figure 1: Comparison between conventional digital systems (left) and the processing-in-memory (PIM) systems (right). Unlike conventional digital systems where quantization is only applied once on both inputs and weights for efficient integer convolution, processing-in-memory (PIM) systems require an **extra quantization** due to **limited resolution** of the analog-to-digital interface. To reduce its effect, the whole convolution is typically decomposed into smaller one and the final results are recombined together. This extra quantization step significantly deteriorates the accuracy performance of model running on it and is the main topic of our work. Note the input to PIM quantization can have a much smaller range than 32-bit integers and here we use "INT32" to denote the most general case.

analog, which is known to be more efficient than in digital for low-precision computation. As an example, a recent PIM demonstration (Yao et al., 2020) achieves 110× higher energy efficiency and 30× better compute density than TESLA V100 GPU. Meanwhile, PIM systems can be built upon various types of integrated memory technologies, from static random-access memory (SRAM) that scales well with Moore's law, to emerging non-volatile memories that stores an analog weight in a tiny unit, e.g. resistive random-access memory (ReRAM) (Prezioso et al., 2015; Xue et al., 2020; Yao et al., 2020) and phase change memory (PCM) (Ambrogio et al., 2018; Joshi et al., 2020).

Despite the forthcoming efficiency and throughput gains, PIM systems require an extra quantization step to digitize the analog MAC results because the high-precision scaling multiplication is more efficient in digital domain (see Fig. 1). However, such extra quantization typically has limited resolution (typically 5-8 bit) due to the hardware constraints and thus leads to significant inference accuracy loss. Furthermore, the inevitable **non-idealilies** in the PIM quantization, including the imperfect linearity and random thermal noise of the analog-to-digital converters (ADCs), aggravate the side-effect of the low-resolution quantization and turn the conventionally quantized model into random guess, as shown in Fig. 2. Limited by the memory array size and analog computing precision, as well as to reduce the input range for less quantization errors, PIM systems compute MACs in a k-bit-serial fashion ($1 \leq k \leq$ input/weight bit-width) and decompose the channels into multiple subsets. The partial sums of PIM output are then re-combined via digital shift-and-adds or accumulation (see Fig. 1). As the computing flow is fundamentally different from conventional models, a new method specialized for PIM systems, taking the **decomposition**, **quantization**, as well as **recombination** into account, is highly desired.

In this paper, we systematically analyze the discrepancies between PIM and conventional digital hardware, and propose *PIM quantization aware training* (**PIM-QAT**) for deep neural networks. Note that in this work we focus on the extra quantization step involved in all types of PIM systems, as mentioned above, and only consider imperfect linearity and stochastic thermal noise. More sophisticated cases of hard-to-model non-linearities caused by inaccurate storage of weights or other effects like data retention issues are out of scope of this work, as they are less general but specific to some types of PIM systems, such as ReRAM. Our method is ideally suitable for the SRAM PIM, where only non-idealities coming from ADCs play a role. However, the problem of PIM quantization is general enough and ubiquitous to all other types of PIM systems, which share the same computing flow as ours, despite their different memory technologies and hardware topologies, including PCM and ReRAM PIMs. Therefore, our method is general and will greatly benefit models running on these systems. We summarize our contributions as the following:

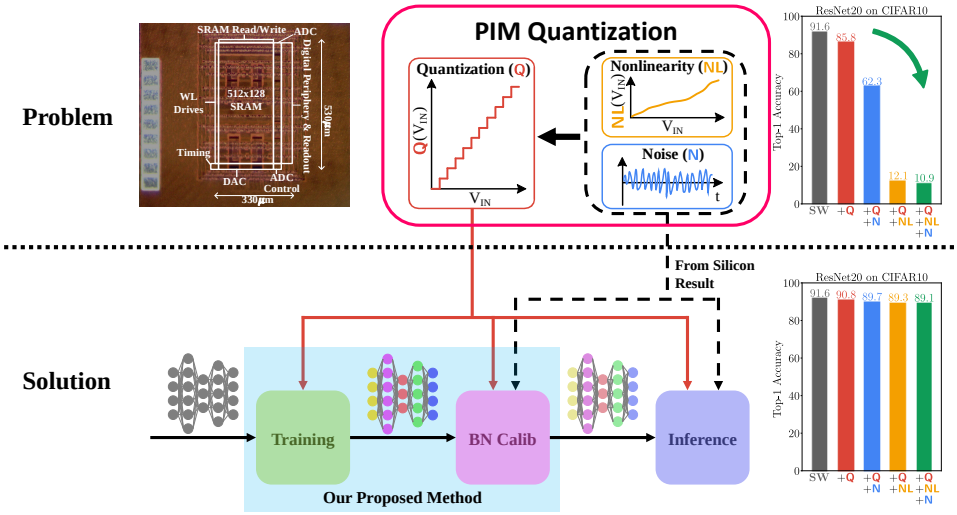

Figure 2: The problem of implementing MAC on a processing in-memory (PIM) system and our proposed solution. Compared to conventional digital systems, the extra low-resolution quantization step in PIM systems introduces significant information loss, making those models trained with conventional quantization techniques fail. The two non-idealities of the extra quantization step, namely imperfect linearity (which we simply denote as non-linearity) and stochastic thermal noise, further aggravate the errors of PIM-quantization. On the contrary, our method takes into account the ideal PIM quantization during training, and applies BN calibration and adjusted precision training algorithm to alleviate the impact of the two non-ideal effects. Note that non-idealities are not directly modeled during training because PIM quantization in different chips exhibits different linearity and noise behaviors due to inter-die variations. Therefore, training with limited non-ideal samples may lead to biased results. Our techniques reduce the accuracy gap and improve the robustness for real PIM systems.

- We propose PIM-QAT based on a basic assumption of *generalized straight-through estimator* (**GSTE**). GSTE is a generalization of the famous straight-through estimator (STE) Bengio et al. (2013), which has been adopted in conventional quantization Zhou et al. (2016).
- We study the *training dynamics* unique to the PIM-QAT, and propose *scaling techniques* for both forward and backward propagation during training to tackle convergence problems.
- We leverage *Batch Normalization (BN) calibration* to close the gap between idealized training and real-case inference on real PIM systems with fabrication and run-time variations.
- We further propose an *adjusted precision training* algorithm and study the potential relations between training precision and the effective number of bits (ENOB) of the actual physical PIM system for inference.
- We test the proposed method on three major PIM decomposition schemes (native, bit serial, differential) that cover the majority of PIM hardware designs. We extensively evaluate the method on a **silicon prototype** of SRAM PIM with realistic non-idealities. A micrograph of the prototype chip is shown in Fig. 2.

2 Background and Related Work

Processing In-Memory Hardware Low-precision PIM quantization is ubiquitous in state-of-the-art PIM systems. Depending on different accuracy targets and model sizes, the quantization resolution ranges from 1-bit (Yin et al., 2020) to 8-bit (Jia et al., 2021), and most of them introduce large quantization errors. The possible levels of the analog MAC results can be up to 67.5× larger than the quantization levels (Lee et al., 2021). On the other hand, different PIM systems adopt different decomposition strategies. The maximum number of elements (N) in one analog MAC is an important parameter because a larger N brings more energy savings, but

Table 1: Energy efficiency of different hardware.

Hardware	V100 GPU	TPU	ReRAM PIM	SRAM (Ours)
Efficiency (TOPS/W)	0.1	2.3	11	49.6

also extends the levels of analog MACs (which is proportional to N). In reality, N is selected from 9 (Yoon et al., 2021) to 2304 (Valavi et al., 2019), making the effect of channel-wise decomposition unique in different PIM systems. Meanwhile, weights are stored in different formats as digital memories (e.g. SRAM) only store 1-bit data in each cell while analog memories (e.g. ReRAM) have multi-state storage, and inputs are decomposed depending on the resolution of digital-to-analog converters (DACs). As a result, different memory topologies lead to different PIM decomposition schemes and quantization errors. Our proposed method unifies all the design choices above and tackles the quantization challenges under various hardware settings.

Despite the potential accuracy loss, PIM is a promising approach for deep learning applications due to its high energy efficiency. Table 1 summarizes the efficiency of V100 GPU (Mujtaba, 2017), TPU (Jouppi et al., 2017), ReRAM PIM (Yao et al., 2020), and our SRAM PIM prototype, which represents "peak" energy efficiency at 100% utilization of the hardware. Training techniques specific for PIM systems is thus an urgent demand.

Analog Computing/PIM Aware Quantization As discussed above, directly applying conventionally trained models on PIM or other analog computing platforms will lead to poor accuracy. To this end, (Joshi et al., 2020) studies the training for PCM-based PIM hardware. However, the method only considers noise and memory retention issues, leaving alone other non-idealities such as quantization and nonlinearity. Also, the evaluation is performed on a simplified system model. Parallel to this work, (Rekhi et al., 2019) proposes a more general analog/mixed signal (AMS) error model, where PIM quantization together with its non-idealities are summarized into an additive noise determined by the effective number of bits (ENOB) of the whole system. Such a high-level abstraction is broadly applicable to different PIM decomposition schemes without considering the detailed implementations, but it also renders sub-optimal results. As shown in Table 2, it is unclear how to estimate ENOB for complex PIM decomposition schemes such as bit serial and differential. Meanwhile, different ENOBs require individually trained models, and the underlying assumption of having a sufficiently large N for central limit theorem does not hold for many practical PIM systems. In this paper, we attempt to solve this discrepancy by incorporating a more interpretable and white-box model for any given PIM hardware in the training procedure.

Table 2: Comparison of training methods for neural networks applied on processing in-memory (PIM) systems.

	Native	Bit Serial	Differential
Baseline	✗	✗	✗
AMS	✓	✗	✗
Ours	✓	✓	✓

3 PIM Quantization Aware Training

In this section, we first describe a generic model of the extra quantization involved in typical PIM systems, and introduce our basic assumption - generalized straight-through estimator (GSTE). Based on these, we propose our PIM-QAT method (Fig. 2), including two scaling techniques to stabilize training dynamics, BN calibration to adapt to fabrication variations of PIM hardware, and an adjusted precision training approach to account for stochastic thermal noise and imperfect linearity together with its chip-to-chip variations.

3.1 Problem Definition

Multiply-and-accumulate (MAC) is the basic operation involved in typical neural networks, including convolution, recurrent, fully-connect, as well as attention layers. Compared to software implementation with a digital system, where the inner product of weight W_i and x_i is given by $y = \sum_{i=1}^{N} W_i x_i$, the output of inner product implemented on a generic PIM system can be formulated as

$$\widetilde{y}_{\text{PIM}} = \mathbf{Q}(\mathbf{NL}(\sum_{i=1}^{N} \widetilde{Q}_i \widetilde{q}_i); b_{\text{PIM}}) + \varepsilon \tag{1}$$

Here, $\widetilde{Q}_i \in [-1, 1]$ and $\widetilde{q}_i \in [0, 1]$ are quantized weights and activations, with b_w and b_a bits, respectively. $\mathbf{Q}$ and $\mathbf{NL}$ denote quantization and imperfect linearity, and ε is the stochastic thermal noise introduced by the system. b_{PIM} is the precision for PIM quantization $\mathbf{Q}$. Eqn. (1) represents

one MAC operation in PIM system (see Analog Computing in Fig. 1), and is generic to different PIM decomposition schemes including native, bit serial, as well as differential schemes (see Sec. 4, also see Appendix A1). Note that the variations of $\widetilde{Q}_i$ and $\widetilde{q}_i$ are not considered here as those non-idealities are only general in analog memories (e.g., ReRAM) but have minor effects in digital memories (e.g., SRAM). We leave this feature as a future investigation.

3.2 Generalized Straight-Through Estimator

In order to take the full advantage of and adapt the neural network to PIM systems, we need to make training aware of $\mathbf{Q}$. For this purpose, we first investigate the conventional quantization-aware training targeting digital accelerators. Generally, in order to back-propagate through a quantized neural network, where the non-differentiable function $\mathrm{round}(\cdot)$ is extensively used, the typical practice is to adopt the straight-through estimator (Bengio et al., 2013) as proposed in (Zhou et al., 2016), where for a real input $r_i \in [0, 1]$, the derivative of quantized output with respect to the input is given by

$$\frac{\partial}{\partial r_i}\Big(\frac{1}{2^k-1}\mathrm{round}\big((2^k-1)r_i\big)\Big) = 1 \tag{2}$$

Here, k is the number of bits for quantization.

To evaluate the effect of $\mathbf{Q}$ involved in PIM systems for both forward and backward propagation, we first generalize the STE result in equation (2) to a stronger yet more flexible assumption, which we name as *generalized straight-through estimator* (GSTE) and is summarized in Assumption 1.

Assumption 1 (Generalized STE) *The differential of the round function is given by*

$$\mathrm{d}\,\mathrm{round}(x) = \xi \cdot \mathrm{d}x \tag{3}$$

where ξ is a scaling factor assigned empirically.

Note that GSTE can also be viewed as a definition for the differential of the discontinuous function $\mathrm{round}(\cdot)$, and equation (2) can be easily derived from it by setting $\xi = 1$. In practice, ξ can be set to different values for different scenarios (for example, for different bit-widths or inputs). We will elaborate more on this point in the following. GSTE will serve as the basis for our whole analysis, and as shown in the appendix, from GSTE we can derive the following theorem for PIM-QAT.

Theorem 1 (PIM Quantization Aware Training) *For ideal PIM systems with PIM decomposition schemes including native, bit serial, as well as differential, where the extra quantization taken into account during forward propagation is ideal without imperfect linearity or noise involved, the backward propagation takes exactly the same form as that for conventional quantization, with only the quantized quantity involved are adjusted accordingly. Specifically, for a PIM system with quantized weight $\widetilde{Q}_i \in [-1, 1]$ of b_w bits and quantized input $\widetilde{q}_i \in [0, 1]$ of b_a bits, the forward and backward propagation are given by*

$$\textbf{Forward}\colon \widetilde{y}_{\mathrm{PIM}} = \mathbf{Q}\Big(\sum_{i=1}^{N}\widetilde{Q}_i\widetilde{q}_i; b_{\mathrm{PIM}}\Big) \tag{4a}$$

$$\textbf{Backward}\colon \mathrm{d}\widetilde{y}_{\mathrm{PIM}} = \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N}\widetilde{Q}_i\widetilde{q}_i\Big) \tag{4b}$$

respectively, where N is the total number of MACs of the inner product and b_{PIM} is PIM bit-width. For conventional quantization with digital accelerator, we have $b_{\mathrm{PIM}} = +\infty$ and the forward propagation is reduced to the typical case of $\widetilde{y} = \sum_{i=1}^{N}\widetilde{Q}_i\widetilde{q}_i$.

Theorem 1 demonstrates that quantization introduced by PIM systems only alters the forward propagation and impacts the calculated values of outputs, but does not change the way of taking derivative over inputs and weights. Additionally, it enables awareness of such quantization during gradient calculation, which is critical for optimization of neural networks targeting PIM systems.

3.3 RESCALING

With Theorem 1, we are ready to incorporate PIM quantization during training. However, this does not guarantee good performance, which also relies on a stable training determined by training dynamics (He et al., 2015; Poole et al., 2016; Schoenholz et al., 2016; Yang and Schoenholz, 2017). In a well-trained model, gradients from different layers should be on the same order to guarantee backward information propagation, in order to avoid gradient exploding/vanishing problems (Bengio et al., 1994; Hochreiter, 1991; Hochreiter et al., 2001; Pascanu et al., 2013). As shown in Appendix A3, PIM quantization has a scale-enlarging effect, especially for low bit-width. To understand the impact of this effect, we first introduce the following theorem.

Theorem 2 (Training Dynamics) *For a neural network composed of repeated blocks, where each block is a sequential of a fully-connected layer, some nonlinear effect (for example, the PIM quantization operation), an extra scaling, a batch normalization layer, and the nonlinear activation $\varphi(\cdot)$, as defined as following*

$$x_i^{(l+1)} = \varphi(y_i^{(l)}) \tag{5a}$$

$$y_i^{(l)} = \gamma_i^{(l)} \frac{z_i^{(l)} - \mu_i^{(l)}}{\sigma_i^{(l)}} + \beta_i^{(l)} \tag{5b}$$

$$z_i^{(l)} = \eta^{(l)} \widetilde{z}_i^{(l)} \tag{5c}$$

$$\widetilde{z}_i^{(l)} = f(W_{ij}^{(l)}, x_j^{(l)}) \sim \rho^{(l)} \sum_{j=1}^{n^{(l)}} W_{ij}^{(l)} x_j^{(l)} \tag{5d}$$

where $x^{(l)}$ is the input to the l-th block, $W^{(l)}$ is the weight matrix of the fully-connected layer, $n^{(l)}$ is the number of input neurons, f represents the nonlinear effect, $\rho^{(l)}$ is introduced to demonstrate the effect of the nonlinearity on the scale of output standard deviation, $\eta^{(l)}$ is an extra scaling factor introduced and explained in the following, and $\gamma^{(l)}$, $\beta^{(l)}$, $\sigma^{(l)}$, $\mu^{(l)}$ are parameters and running statistics of the batch norm layer. If the differential of the nonlinear effect f is given by

$$\mathrm{d}\widetilde{z}_i^{(l)} = \xi^{(l)} \cdot \mathrm{d}\left(\sum_{j=1}^{n^{(l)}} W_{ij}^{(l)} x_j^{(l)} \right) \tag{6}$$

where $\xi^{(l)}$ is the scaling factor for backward propagation inside the l-th layer, then for zeroth order approximation (mean-field assumption), the activation gradient variance ratio between two adjacent layers is given by

$$\frac{\mathbb{VAR}[\partial_{x^{(l)}} L]}{\mathbb{VAR}[\partial_{x^{(l+1)}} L]} \approx \left(\frac{\xi^{(l)}}{\rho^{(l)}} \right)^2 \cdot \frac{n^{(l+1)}}{n^{(l)}} \tag{7}$$

Theorem 2 indicates that the scale ratio between activation gradients from two adjacent layers depends on the scaling factors introduced during forward and backward for the nonlinear effect.

Based on the results in (7), we can find that if we do not introduce extra scaling factor as in (6) but follow the conventional practice of STE (in other words, $\xi^{(l)} = 1$ for all l), the scale-enlarging effect may cause gradient exploding/vanishing problem. Proper intialization as proposed in He et al. (2015) is not effective in this case. Experiment demonstrates that for some PIM decomposition scheme (such as bit serial and differential) and sufficiently low bit-width (such as 7-bit), the training does not converge.

To overcome this problem, we propose to scale the gradient according to (6), and determine the necessary scale by also calculating the standard deviation of the result from software quantization. Specifically, the scaling factor in (6) is given by

$$\xi = \sqrt{\frac{\mathbb{VAR}[y_{\mathrm{PIM}}]}{\mathbb{VAR}[y]}} \tag{8}$$

where y_{PIM} is the result with PIM system and y is that with conventional software. Note that this only introduces extra computation during training as the scale factor is only necessary for backward to stablize training, and will not impact the inference procedure. Experiment demonstrates that this backward scaling solves the problem for cases those otherwise do not give reasonable results.

Besides scaling for backward, we find that scaling during forward with predefined constant factor helps the training procedure, especially for low bit-widths, such as those lower than 5-bit. Even for

higher precision, introducing extra scaling can still be beneficial. However, as shown in equation (7), the ratio does not depend on this factor, as it should be absorbed into the running variance of the following batch normalization layers. We guess this is related to numerical stability for computation, but the underlying mechanism is still unclear to us and we leave it as a future work. However, we list the scaling factor that we find best for practice in the appendix.

3.4 BN CALIBRATION

In the above we discuss about PIM systems with ideal quantization, where the extra PIM quantization step is perfectly linear without stochastic thermal noise. For real systems, there are two non-ideal effects. First, the circuit non-idealities in the analog-to-digital conversion will degrade the quantization linearity. Second, random fluctuations in the circuit will add thermal noise on the quantized output. Moreover, the imperfectness accompanying the linear mapping varies from chip to chip, and there lacks a unified model to describe such variation accurately. On the other hand, direct training with injected noise can either deteriorate the training progress (for example, if the noise injected is too large), or the noise energy can be different for different real systems. Consequently, it is almost impossible to directly consider these effects during training, especially in backward propagation.

Experiments demonstrate that the non-idealities have the potential to change the BN statistics (see Appendix A4), and following (Yu and Huang, 2019), we propose to use a small portion of training data and calibrate BN running statistics before evaluation. For both BN calibration and final inference, we apply exactly the same real-case non-idealities. We find this can significantly improve the performance, especially when the non-ideal effect is strong (e.g., more imperfect linearity or larger injected noise). Note that Joshi et al. (2020) exploits calibrating batch normalization statistics for the purpose of accuracy retention involved in PCM systems, which is a different problem from ours. In Appendix A7, we present more experiments, where we find that BN calibration is able to reduce the impact of gain and offset in PIM quantization and thus alleviates hardware calibration efforts.

3.5 ADJUSTED PRECISION TRAINING

Besides BN calibration, we study the possibility of employing different precisions for training and inference. The reasoning behind is that the non-idealities only affect the least significant bits during the involved quantization mapping, which effectively reduces the number of distinguishable output levels from the PIM system. To quantify this reduction, ADC designs typically use a metric called the effective number of bit-width (ENOB) and it can be adopted here. As an example, Fig. 3 shows that the standard deviation of MAC computing errors in a 7-bit PIM system will be equal to that of ideal lower bit PIM systems, when random noise is added. Note that this adjusted precision training method considers both noise injection and imperfect linearity. Depending on quantization bit-widths, noise levels, imperfect linearity forms, the optimal training precision varies but is expected to be always smaller than the ideal PIM resolution.

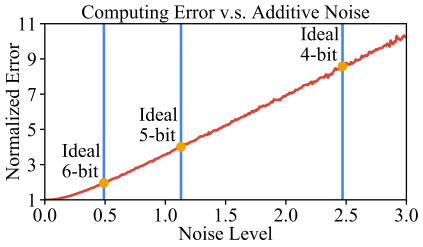

Figure 3: Computing error as a function of the standard deviation of additive noise in our 7-bit PIM chip.

4 EXPERIMENTS

Native Scheme. We first investigate the possibility of directly applying the conventional quantized model on PIM system, which serves as the baseline for our comparison. To this end, we take the native scheme as an example, and fix the number of multiplications for each processing to 9, namely we use a unit channel of 1 to split the input channels. We experiment on CIFAR10 with ResNet20, and the results are summarized in Table 3. Our method significantly outperforms baseline, especially for ultra-low bit-widths. As shown in Table 3, the AMS method in (Rekhi et al., 2019) is supposed to work for the native scheme. It indeed improves over the baseline but shows inferior performance than ours. These results demonstrate that PIM quantization has non-negligible impacts

Table 4: Accuracy with the 7-bit real chip (with the real chip curve from Figure A1 and noise level of 0.35) of bit-serial PIM system for different datasets and models. Note that PIM systems is hundreds times more efficient than software system.

Dataset	Depth	Method	N	Acc.↑	Efficiency (TOPS/W)↑	Depth	Method	N	Acc.↑	Efficiency (TOPS/W)↑
CIFAR10	20	Software	-	91.6	0.1	44	Software	-	92.8	0.1
		Baseline	72	13.9	**24.8**		Baseline	72	10.5	**24.8**
			144	10.9	**49.6**			144	10.0	**49.6**
		Ours	72	**89.7**	**24.8**		**Ours**	72	**90.6**	**24.8**
			144	**89.1**	**49.6**			144	**90.7**	**49.6**
	32	Software	-	92.5	0.1	56	Software	-	92.4	0.1
		Baseline	72	10.0	**24.8**		Baseline	72	10.0	**24.8**
			144	10.1	**49.6**			144	10.0	**49.6**
		Ours	72	**90.6**	**24.8**		**Ours**	72	**90.7**	**24.8**
			144	**89.3**	**49.6**			144	**90.4**	**49.6**
CIFAR100	20	Software	-	67.0	0.1	56	Software	-	70.3	0.1
		Baseline	72	1.8	**24.8**		Baseline	72	1.0	**24.8**
			144	1.3	**49.6**			144	1.1	**49.6**
		Ours	72	**62.6**	**24.8**		**Ours**	72	**65.3**	**24.8**
			144	**61.8**	**49.6**			144	**63.5**	**49.6**

on the final accuracy, and it is necessary to take this quantization into account during training for optimal inference accuracy on PIM systems.

Real Chip Results. We experiment on CIFAR10 and CIFAR100 datasets, with several ResNet models. We also use different numbers of unit channels, namely 8 and 16, to split the input channels, corresponding to number of computing units of 72 and 144, respectively. As shown in Table 4, our method provides significantly better results than the baseline. Specifically, prediction in the baseline models is barely better than random guess, meaning the non-idealities from the real chip corrupt the behavior of neural networks trained in this way. In contrast, our method gives comparable results as those on digital system (the software results), meaning the trained models are robust to real-case non-idealities. Note that using smaller N typically leads to better performance, especially for CIFAR100, at the cost of reduced throughput and energy efficiency.

Table 3: Effect of PIM quantization on accuracy of neural networks trained with different methods for native scheme ($N = 9$). Baseline refers to model trained with conventional quantization method (Jin et al., 2020). AMS refers to model trained with the method in (Rekhi et al., 2019).

b_{PIM}	Method	Acc.
3	Baseline	8.3
	AMS	73.3
	Ours	**81.7**
4	Baseline	27.2
	AMS	85.0
	Ours	**87.7**
5	Baseline	80.5
	AMS	89.0
	Ours	**90.7**
6	Baseline	89.2
	AMS	90.3
	Ours	**90.9**
7	Baseline	**91.0**
	AMS	90.7
	Ours	**91.0**
$+\infty$	Baseline	91.6

Other PIM Decomposition Schemes. We further verify our method on three other most common PIM decomposition schemes, including native, differential and bit serial (see Appendix A1). We experiment on ideal PIM with different inference resolutions and noise levels. As shown in Figure 4, we compare our method with the baseline using BN calibration on ResNet20 with CIFAR10 dataset. It is clear that for all schemes with different resolution and noise levels, our method is consistently superior, especially for high noise level and for differential and bit-serial schemes, both of which are more practical and complex than the native one. This justifies that our proposed method is applicable to a wide range of PIM implementations.

Adjusted Precision Training. Here we provide some ablation studies on adjusted precision training. We use an ideal PIM system with bit serial scheme as the example. For different inference resolutions and noise levels, the best accuracy with the optimal training resolution is illustrated in Fig. 5, where the accuracy is directly listed and different colors denote different training precision adjustments. We find that for low noise level, it is optimal to train the model with the same resolution as that for inference, and for larger noise, it is better to use a smaller one due to reduced ENOB.

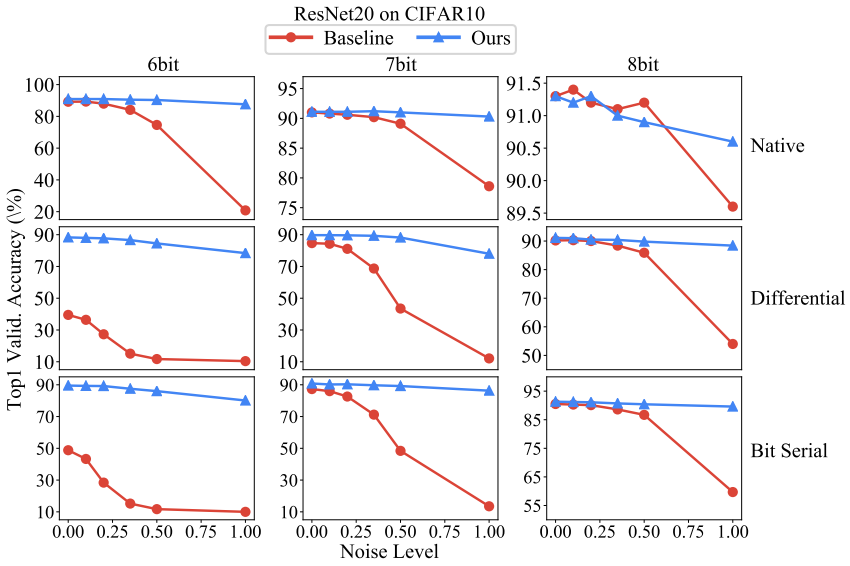

Figure 4: Performance of ResNet20 on CIFAR10 with ideal PIM of different schemes and PIM resolutions. Note that $N = 9$ for native and $N = 144$ for differential and bit serial schemes.

Moreover, we find that the noise level threshold of adjusting the training resolution depends on the absolute value of inference resolution, and higher inference resolution tends to be more sensitive to noise and requires precision adjustment for a lower noise level threshold. There is clear correlation between the precision reduction and ENOB, but they are not exactly the same. This should be related to the varying sensitivity of inference on each MAC operation. More in-depth analysis of the relation between ENOB and training setting is beyond the scope of this work and left for future study.

Our analysis and experiments demonstrate that naively deploying neural network quantized with conventional method on PIM systems is problematic and ineffective, and PIM quantization has non-negligible impact on final performance. Incorporating it into training is critical and will improve the accuracy to a large extent. It also inspires and provides a desirable starting point for future research to incorporate hardware-specific behaviors into algorithm co-design for energy-efficient analog computing systems. Such efforts will bridge the gap between hardware and software developments to achieve unprecedented energy efficiency, while maintaining a competitive neural network performance.

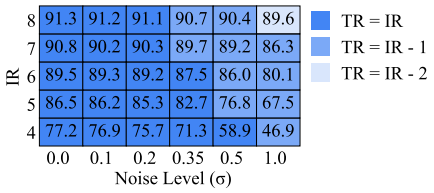

Figure 5: The desirable training resolutions (TR) for different inference resolutions (IR) and noise levels, with bit-serial scheme.

5 Conclusion

In this paper, we systematically study the problem of training a neural network for application on the processing in-memory (PIM) system, which is a promising candidate for next-generation hardware for deep learning, and we provide a method for the extra quantization step unique to PIM systems but ubiquitous to all different types of PIM implementations. Specifically, we formulate the problem and analyze the forward and backward propagation to enable PIM quantization-aware training. We study the training dynamics of our method, and propose rescaling techniques for both forward and backward propagations, to avoid gradient exploding/vanishing issues. We also study the discrepancy between training and inference, where more realistic non-ideal effects, such as imperfect linearity and stochastic thermal noise, are involved but difficult to incorporate during backward propagation. To this end, we propose to leverage BN calibration technique and invent adjusted precision training. Finally, we present experimental results to demonstrate potential relationship between training and inference bit-widths, together with noise level and effective number of bits for the PIM system for inference.

REFERENCES

Ambrogio, S., Narayanan, P., Tsai, H., Shelby, R. M., Boybat, I., Di Nolfo, C., Sidler, S., Giordano, M., Bodini, M., Farinha, N. C., et al. (2018). Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558(7708):60–67.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., et al. (2014). Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE.

Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. (2016). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138.

Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I.-J., Srinivasan, V., and Gopalakrishnan, K. (2018). Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*.

Du, Z., Fasthuber, R., Chen, T., Ienne, P., Li, L., Luo, T., Feng, X., Chen, Y., and Temam, O. (2015). ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 92–104.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1).

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Horowitz, M. (2014). 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE.

Ielmini, D. and Wong, H.-S. P. (2018). In-memory computing with resistive switching devices. *Nature Electronics*, 1(6):333–343.

Jia, H., Ozatay, M., Tang, Y., Valavi, H., Pathak, R., Lee, J., and Verma, N. (2021). 15.1 a programmable neural-network inference accelerator based on scalable in-memory computing. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 236–238. IEEE.

Jia, H., Valavi, H., Tang, Y., Zhang, J., and Verma, N. (2020). A programmable heterogeneous microprocessor based on bit-scalable in-memory computing. *IEEE Journal of Solid-State Circuits*, pages 1–1.

Jiang, Z., Yin, S., Seo, J.-S., and Seok, M. (2019). C3SRAM: In-memory-computing SRAM macro based on capacitive-coupling computing. *IEEE Solid-State Circuits Letters*, 2(9):131–134.

Jin, Q., Yang, L., and Liao, Z. (2019). Towards efficient training for neural network quantization. *arXiv preprint arXiv:1912.10207*.

Jin, Q., Yang, L., Liao, Z., and Qian, X. (2020). Neural network quantization with scale-adjusted training. In *The 31st British Machine Vision Conference (BMVC)*.

Joshi, V., Le Gallo, M., Haefeli, S., Boybat, I., Nandakumar, S. R., Piveteau, C., Dazzi, M., Rajendran, B., Sebastian, A., and Eleftheriou, E. (2020). Accurate deep neural network inference using computational phase-change memory. *Nature communications*, 11(1):1–13.

Jouppi, N. P. et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12.

Lee, J., Valavi, H., Tang, Y., and Verma, N. (2021). 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*, pages 1–2. IEEE.

Liu, S., Ren, B., Shen, X., and Wang, Y. (2020). Cocopie: Making mobile ai sweet as pie–compression-compilation co-design goes a long way. *arXiv preprint arXiv:2003.06700*.

Mujtaba, H. (2017). Nvidia volta gv100 12nm finfet gpu detailed – tesla v100 specifications include 21 billion transistors, 5120 cuda cores, 16 gb hbm2 with 900 gb/s bandwidth. *Wccftech*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR.

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. (2016). Exponential expressivity in deep neural networks through transient chaos. *arXiv preprint arXiv:1606.05340*.

Prezioso, M., Merrikh-Bayat, F., Hoskins, B. D., Adam, G. C., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61–64.

Rekhi, A. S., Zimmer, B., Nedovic, N., Liu, N., Venkatesan, R., Wang, M., Khailany, B., Dally, W. J., and Gray, C. T. (2019). Analog/mixed-signal hardware error modeling for deep learning inference. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6.

Schoenholz, S. S., Gilmer, J., Ganguli, S., and Sohl-Dickstein, J. (2016). Deep information propagation. *arXiv preprint arXiv:1611.01232*.

Si, X., Tu, Y.-N., Huanq, W.-H., Su, J.-W., Lu, P.-J., Wang, J.-H., Liu, T.-W., Wu, S.-Y., Liu, R., Chou, Y.-C., Zhang, Z., Sie, S.-H., Wei, W.-C., Lo, Y.-C., Wen, T.-H., Hsu, T.-H., Chen, Y.-K., Shih, W., Lo, C.-C., Liu, R.-S., Hsieh, C.-C., Tang, K.-T., Lien, N.-C., Shih, W.-C., He, Y., Li, Q., and Chang, M.-F. (2020). 15.5 A 28nm 64Kb 6T SRAM computing-in-memory macro with 8b MAC operation for AI edge chips. In *2020 IEEE International Solid- State Circuits Conference (ISSCC)*, pages 246–248.

Su, J.-W., Si, X., Chou, Y.-C., Chang, T.-W., Huang, W.-H., Tu, Y.-N., Liu, R., Lu, P.-J., Liu, T.-W., Wang, J.-H., Zhang, Z., Jiang, H., Huang, S., Lo, C.-C., Liu, R.-S., Hsieh, C.-C., Tang, K.-T., Sheu, S.-S., Li, S.-H., Lee, H.-Y., Chang, S.-C., Yu, S., and Chang, M.-F. (2020). 15.2 A 28nm 64Kb Inference-Training Two-Way Transpose Multibit 6T SRAM Compute-in-Memory Macro for AI Edge Chips. In *2020 IEEE International Solid- State Circuits Conference (ISSCC)*, pages 240–242.

Valavi, H., Ramadge, P. J., Nestler, E., and Verma, N. (2019). A 64-Tile 2.4-Mb In-Memory-Computing CNN Accelerator Employing Charge-Domain Compute. *IEEE Journal of Solid-State Circuits*, 54(6):1789–1799.

Xue, C.-X. et al. (2020). A cmos-integrated compute-in-memory macro based on resistive random-access memory for ai edge devices. *Nature Electronics*, pages 1–10.

Yang, G. and Schoenholz, S. S. (2017). Mean field residual networks: On the edge of chaos. *arXiv preprint arXiv:1712.08969*.

Yao, P., Wu, H., Gao, B., Tang, J., Zhang, Q., Zhang, W., Yang, J. J., and Qian, H. (2020). Fully hardware-implemented memristor convolutional neural network. *Nature*, 577(7792):641–646.

Ye, S., Zhang, T., Zhang, K., Li, J., Xu, K., Yang, Y., Yu, F., Tang, J., Fardad, M., Liu, S., et al. (2018). Progressive weight pruning of deep neural networks using admm. *arXiv preprint arXiv:1810.07378*.

Yin, S., Jiang, Z., Seo, J.-S., and Seok, M. (2020). XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits*.

Yoon, J.-H., Chang, M., Khwa, W.-S., Chih, Y.-D., Chang, M.-F., and Raychowdhury, A. (2021). A 40-nm, 64-Kb, 56.67 TOPS/W voltage-sensing computing-in-memory/digital RRAM macro supporting iterative write with verification and online read-disturb detection. *IEEE Journal of Solid-State Circuits*.

Yu, J. and Huang, T. S. (2019). Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1803–1811.

Yue, J., Yuan, Z., Feng, X., He, Y., Zhang, Z., Si, X., Liu, R., Chang, M.-F., Li, X., Yang, H., and Liu, Y. (2020). 14.3 A 65nm computing-in-memory-based CNN processor with 2.9-to-35.8TOPS/W system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse. In *2020 IEEE International Solid- State Circuits Conference (ISSCC)*, pages 234–236.

Zhang, J., Wang, Z., and Verma, N. (2017). In-memory computation of a machine-learning classifier in a standard 6T SRAM array. *IEEE Journal of Solid-State Circuits*, 52(4):915–924.

Zhang, T., Ye, S., Zhang, K., Tang, J., Wen, W., Fardad, M., and Wang, Y. (2018). A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199.

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

Appendix

A1 PROOF OF THEOREMS

Here we present detailed proofs for Theorem 1 and 2. We first formulate the quantization procedure of PIM systems with several popular schemes, then derive the results of Theorem 1. After that, we analyze the training dynamics of a generic neural networks to prove Theorem 2.

A1.1 PIM QUANTIZATION-AWARE TRAINING

To prove Theorem 1, we first present the quantization procedure of PIM systems with native, differential and bit serial schemes. The output of a linear layer is given by

$$\widetilde{y} = \sum_{i=1}^{N} \widetilde{Q}_i \widetilde{q}_i \tag{A1}$$

where $\widetilde{Q}_i \in [-1, 1]$ is quantized weight of b_w bits and $\widetilde{q}_i \in [0, 1]$ is quantized input of b_a bits, respectively.

For PIM systems, due to the limited resolution of digital-to-analog converter (which is m bits) for the inputs, the inputs are first decomposed into sub-arrays of m bits. In other words, we have

$$\widetilde{q}_i = \sum_{k=0}^{b_a/m-1} \widetilde{q}_{i,k}^{(m)} \Delta^k \tag{A2a}$$

$$\widetilde{q}_{i,k}^{(m)} = \frac{1}{2^{b_a}-1} q_{i,k}^{(m)} \tag{A2b}$$

$$\Delta = 2^m \tag{A2c}$$

with $q_{i,k}^{(m)} \in \{0, 1, \ldots, \Delta - 1\}$.

Native Scheme For PIM system with native scheme, the output is given by

$$\widetilde{y}_{\text{PIM}} = \mathbf{Q}\Big(\sum_{i=1}^{N} \widetilde{Q}_i \widetilde{q}_i; b_{\text{PIM}} \Big) \tag{A3a}$$

$$= \sum_{k=0}^{b_a/m-1} \Delta^k \cdot \frac{N(\Delta-1)}{(2^{b_{\text{PIM}}}-1)(2^{b_a}-1)} \cdot \text{round}\Big(\frac{(2^{b_{\text{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_i \cdot \widetilde{q}_{i,k}^{(m)} \Big) \tag{A3b}$$

With the GSTE assumption, we can derive its differential as

$$\mathrm{d}\widetilde{y}_{\text{PIM}} = \sum_{k=0}^{b_a/m-1} \Delta^k \cdot \frac{N(\Delta-1)}{(2^{b_{\text{PIM}}}-1)(2^{b_a}-1)} \cdot \mathrm{d}\Big[\text{round}\Big(\frac{(2^{b_{\text{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_i \cdot \widetilde{q}_{i,k}^{(m)} \Big) \Big] \tag{A4a}$$

$$= \sum_{k=0}^{b_a/m-1} \Delta^k \cdot \frac{N(\Delta-1)}{(2^{b_{\text{PIM}}}-1)(2^{b_a}-1)} \cdot \xi \cdot \mathrm{d}\Big(\frac{(2^{b_{\text{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_i \cdot \widetilde{q}_{i,k}^{(m)} \Big) \tag{A4b}$$

$$= \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} \widetilde{Q}_i \cdot \sum_{k=0}^{b_a/m-1} \widetilde{q}_{i,k}^{(m)} \Delta^k \Big) \tag{A4c}$$

$$= \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} \widetilde{Q}_i \widetilde{q}_i \Big) \tag{A4d}$$

Differential Scheme For PIM system with differential scheme, the weight is first decomposed into positive and negative parts, as

$$\widetilde{Q}_i = \widetilde{Q}_i^+ + \widetilde{Q}_i^- \tag{A5}$$

where all elements in $\widetilde{Q}_i^+$ are positive and those in $\widetilde{Q}_i^-$ are negative. Its differential is given by

$$\mathrm{d}\widetilde{Q}_i = \mathrm{d}\widetilde{Q}_i^+ + \mathrm{d}\widetilde{Q}_i^- \tag{A6}$$

The output is the combination of these two parts as

$$\widetilde{y}_{\mathrm{PIM}} = \mathbf{Q}\Big(\sum_{i=1}^{N} \widetilde{Q}_i \widetilde{q}_i; b_{\mathrm{PIM}}\Big) \tag{A7a}$$

$$= \sum_{k=0}^{b_a/m-1} \Delta^k \cdot \frac{N(\Delta-1)}{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)} \cdot \Big[\mathrm{round}\Big(\frac{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_i^+ \widetilde{q}_{i,k}^{(m)}\Big) - \mathrm{round}\Big(\frac{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} (-\widetilde{Q}_i^-) \widetilde{q}_{i,k}^{(m)}\Big)\Big] \tag{A7b}$$

Taking differential on both sides gives

$$\mathrm{d}\widetilde{y}_{\mathrm{PIM}} = \mathrm{d} \sum_{k=0}^{b_a/m-1} \Delta^k \cdot \frac{N(\Delta-1)}{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)} \cdot \Big[\mathrm{round}\Big(\frac{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_i^+ \widetilde{q}_{i,k}^{(m)}\Big) - \mathrm{round}\Big(\frac{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} (-\widetilde{Q}_i^-) \widetilde{q}_{i,k}^{(m)}\Big)\Big] \tag{A8a}$$

$$= \sum_{k=0}^{b_a/m-1} \Delta^k \cdot \frac{N(\Delta-1)}{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)} \cdot \Big\{\mathrm{d}\Big[\mathrm{round}\Big(\frac{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_i^+ \widetilde{q}_{i,k}^{(m)}\Big)\Big] - \mathrm{d}\Big[\mathrm{round}\Big(\frac{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} (-\widetilde{Q}_i^-) \widetilde{q}_{i,k}^{(m)}\Big)\Big]\Big\} \tag{A8b}$$

$$= \sum_{k=0}^{b_a/m-1} \Delta^k \cdot \frac{N(\Delta-1)}{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)} \cdot \Big[\xi \cdot \mathrm{d}\Big(\frac{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_i^+ \widetilde{q}_{i,k}^{(m)}\Big) - \xi \cdot \mathrm{d}\Big(\frac{(2^{b_{\mathrm{PIM}}}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} (-\widetilde{Q}_i^-) \widetilde{q}_{i,k}^{(m)}\Big)\Big] \tag{A8c}$$

$$= \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} \widetilde{Q}_i^+ \sum_{k=0}^{b_a/m-1} \widetilde{q}_{i,k}^{(m)} \Delta^k\Big) + \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} \widetilde{Q}_i^- \sum_{k=0}^{b_a/m-1} \widetilde{q}_{i,k}^{(m)} \Delta^k\Big) \tag{A8d}$$

$$= \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} \widetilde{Q}_i^+ \widetilde{q}_i\Big) + \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} \widetilde{Q}_i^- \widetilde{q}_i\Big) \tag{A8e}$$

$$= \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} (\widetilde{Q}_i^+ + \widetilde{Q}_i^-) \widetilde{q}_i\Big) \tag{A8f}$$

$$= \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} \widetilde{Q}_i \widetilde{q}_i\Big) \tag{A8g}$$

Bit Serial Scheme For PIM system with bit serial scheme, the weight is first decomposed into bits as

$$\widetilde{Q}_i = \sum_{k=0}^{b_w-1} (-1)^{\delta_{k,b_w-1}} \widetilde{Q}_{i,k} 2^k \tag{A9}$$

where

$$\widetilde{Q}_{i,k} = \frac{1}{2^{b_w-1}-1} Q_{i,k} \tag{A10}$$

and $Q_{i,k} \in \{0, 1\}$. The output is obtained for each bits separately and then summed over as

$$\widetilde{y}_{\text{PIM}} = \mathbf{Q}\Big(\sum_{i=1}^{N} \widetilde{Q}_i \widetilde{q}_i; b_{\text{PIM}}\Big) \tag{A11a}$$

$$= \sum_{k=0}^{b_w-1} \sum_{l=0}^{b_a/m-1} (-1)^{\delta_{k,b_w-1}} 2^k \Delta^l \cdot \frac{N(\Delta-1)}{(2^{b_{\text{PIM}}}-1)(2^{b_w-1}-1)(2^{b_a}-1)}$$

$$\cdot \text{round}\Big(\frac{(2^{b_{\text{PIM}}}-1)(2^{b_w-1}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_{i,k} \widetilde{q}_{i,l}^{(m)}\Big) \tag{A11b}$$

The differential can thus be determined as

$$\mathrm{d}\widetilde{y}_{\text{PIM}} = \mathrm{d} \sum_{k=0}^{b_w-1} \sum_{l=0}^{b_a/m-1} (-1)^{\delta_{k,b_w-1}} 2^k \Delta^l \cdot \frac{N(\Delta-1)}{(2^{b_{\text{PIM}}}-1)(2^{b_w-1}-1)(2^{b_a}-1)}$$

$$\cdot \text{round}\Big(\frac{(2^{b_{\text{PIM}}}-1)(2^{b_w-1}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_{i,k} \widetilde{q}_{i,l}^{(m)}\Big) \tag{A12a}$$

$$= \sum_{k=0}^{b_w-1} \sum_{l=0}^{b_a/m-1} (-1)^{\delta_{k,b_w-1}} 2^k \Delta^l \cdot \frac{N(\Delta-1)}{(2^{b_{\text{PIM}}}-1)(2^{b_w-1}-1)(2^{b_a}-1)}$$

$$\cdot \mathrm{d}\Big[\text{round}\Big(\frac{(2^{b_{\text{PIM}}}-1)(2^{b_w-1}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_{i,k} \widetilde{q}_{i,l}^{(m)}\Big)\Big] \tag{A12b}$$

$$= \sum_{k=0}^{b_w-1} \sum_{l=0}^{b_a/m-1} (-1)^{\delta_{k,b_w-1}} 2^k \Delta^l \cdot \frac{N(\Delta-1)}{(2^{b_{\text{PIM}}}-1)(2^{b_w-1}-1)(2^{b_a}-1)}$$

$$\cdot \xi \cdot \mathrm{d}\Big(\frac{(2^{b_{\text{PIM}}}-1)(2^{b_w-1}-1)(2^{b_a}-1)}{N(\Delta-1)} \sum_{i=1}^{N} \widetilde{Q}_{i,k} \widetilde{q}_{i,l}^{(m)}\Big) \tag{A12c}$$

$$= \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} \sum_{k=0}^{b_w-1} (-1)^{\delta_{k,b_w-1}} \widetilde{Q}_{i,k} 2^k \cdot \sum_{l=0}^{b_a/m-1} \widetilde{q}_{i,l}^{(m)} \Delta^l\Big) \tag{A12d}$$

$$= \xi \cdot \mathrm{d}\Big(\sum_{i=1}^{N} \widetilde{Q}_i \widetilde{q}_i\Big) \tag{A12e}$$

With (A4d), (A8g) and (A12e), we prove Theorem 1.

A1.2 Training Dynamics

Here we analyze the training dynamics to prove Theorem 2. Our analysis is similar to that presented in (Jin et al., 2019), which is zeroth order approximation and is based on mean field theory, where different quantities are assumed to be independent (although some of them have some dependence, especially the gradients, as described following Axiom 3.2 in (Yang and Schoenholz, 2017)).

We want to analyze the training dynamics of a generic neural network with repeated blocks composed of linear layer, some nonlinear effect, forward scaling, batch normalization, and output activation function, as

$$x_i^{(l+1)} = \varphi(y_i^{(l)}) \tag{A13a}$$

$$y_i^{(l)} = \gamma_i^{(l)} \frac{z_i^{(l)} - \mu_i^{(l)}}{\sigma_i^{(l)}} + \beta_i^{(l)} \tag{A13b}$$

$$z_i^{(l)} = \eta^{(l)} \widetilde{z}_i^{(l)} \tag{A13c}$$

$$\widetilde{z}_i^{(l)} = f(W_{ij}^{(l)}, x_j^{(l)}) \tag{A13d}$$

$$\sim \rho^{(l)} \sum_{j=1}^{n^{(l)}} W_{ij}^{(l)} x_j^{(l)} \tag{A13e}$$

$$\mathrm{d}\tilde{z}_i^{(l)} = \xi^{(l)} \cdot \mathrm{d}\left(\sum_{j=1}^{n^{(l)}} W_{ij}^{(l)} x_j^{(l)} \right) \tag{A13f}$$

where $\rho^{(l)}$ denotes modification on the output variance by the nonlinear effect and $\xi^{(l)}$ is the scaling factor introduced during backward propagation. From this we can derive the following statistics

$$\mathbb{VAR}[y_i^{(l)}] \approx (\gamma_i^{(l)})^2 \tag{A14a}$$

$$\mathbb{E}[(x_i^{(l)})^2] \approx \mathbb{E}[(\varphi'(y_i^{(l)}))^2]\mathbb{VAR}[y_i^{(l)}] \tag{A14b}$$

$$\approx \mathbb{E}[(\varphi'(y_i^{(l)}))^2](\gamma_i^{(l)})^2 \tag{A14c}$$

$$(\sigma_i^{(l)})^2 = \mathbb{VAR}[z_i^{(l)}] \tag{A14d}$$

$$= (\eta^{(l)})^2(\rho^{(l)})^2 n^{(l)} \mathbb{VAR}[W_{ij}^{(l)}]\mathbb{E}[(x_j^{(l)})^2] \tag{A14e}$$

$$\approx (\eta^{(l)})^2(\rho^{(l)})^2 n^{(l)} \mathbb{VAR}[W_{ij}^{(l)}]\mathbb{E}[(\varphi'(y_j^{(l)}))^2](\gamma_j^{(l)})^2 \tag{A14f}$$

where (A14b) is valid if the activation ϕ is quasi-linear.

We first estimate the gradient of the batch normalization layer as following

$$\frac{\partial \mu_i^{(l)}}{\partial z_i^{(l)}} = \frac{1}{m_B} \tag{A15a}$$

$$\frac{\partial \sigma_i^{(l)}}{\partial z_i^{(l)}} = \frac{1}{m_B} \frac{z_i^{(l)} - \mu_i^{(l)}}{\sigma_i^{(l)}} \tag{A15b}$$

$$\frac{\partial y_i^{(l)}}{\partial z_i^{(l)}} = \gamma_i^{(l)} \left[\frac{1}{\sigma_i^{(l)}} - \frac{1}{\sigma_i^{(l)}} \frac{\partial \mu_i^{(l)}}{\partial z_i^{(l)}} + (z_i^{(l)} - \mu_i^{(l)}) \frac{-1}{(\sigma_i^{(l)})^2} \frac{\partial \sigma_i^{(l)}}{\partial z_i^{(l)}} \right] \tag{A15c}$$

$$= \gamma_i^{(l)} \left[\frac{1}{\sigma_i^{(l)}} - \frac{1}{\sigma_i^{(l)}} \frac{1}{m_B} - \frac{1}{\sigma_i^{(l)}} \frac{1}{m_B} \left(\frac{z_i^{(l)} - \mu_i^{(l)}}{\sigma_i^{(l)}} \right)^2 \right] \tag{A15d}$$

$$\approx \frac{\gamma_i^{(l)}}{\sigma_i^{(l)}} \quad (m_B >> 1) \tag{A15e}$$

where we have assumed that the batch size is large enough, which is typically satisfied in practice.

The gradients of loss L with respect to the input can be easily calculated, which is

$$\partial_{x_j^{(l)}} L = \xi^{(l)} \sum_{i=1}^{n^{(l+1)}} W_{ij}^{(l)} \eta^{(l)} \frac{\gamma_i^{(l)}}{\sigma_i^{(l)}} \varphi'(y_i^{(l)}) \partial_{x_i^{(l+1)}} L \tag{A16}$$

from which we can derive the variance of the gradient, based on mean field assumption, as

$$\mathbb{VAR}[\partial_{x_j^{(l)}} L] = n^{(l+1)}(\xi^{(l)})^2(\eta^{(l)})^2 \left(\frac{\gamma_i^{(l)}}{\sigma_i^{(l)}} \right)^2 \mathbb{VAR}[W_{ij}^{(l)}]\mathbb{E}[(\varphi'(y_i^{(l)}))^2]\mathbb{VAR}[\partial_{x_i^{(l+1)}} L] \tag{A17}$$

Substituting (A14f), we have

$$\mathbb{VAR}[\partial_{x_j^{(l)}} L] = n^{(l+1)}(\xi^{(l)})^2 \cancel{(\eta^{(l)})^2} \frac{\cancel{(\gamma_i^{(l)})^2}}{n^{(l)} \cancel{(\eta^{(l)})^2} (\rho^{(l)})^2 \cancel{\mathbb{VAR}[W_{ij}^{(l)}]} \cancel{\mathbb{E}[(\varphi'(y_j^{(l)}))^2]} \cancel{(\gamma_j^{(l)})^2}} \cancel{\mathbb{VAR}[W_{ij}^{(l)}]}$$
$$\cdot \cancel{\mathbb{E}[(\varphi'(y_i^{(l)}))^2]} \mathbb{VAR}[\partial_{x_i^{(l+1)}} L]$$

$$= \left(\frac{\xi^{(l)}}{\rho^{(l)}}\right)^2 \cdot \frac{n^{(l+1)}}{n^{(l)}} \cdot \mathbb{VAR}[\partial_{x_i^{(l+1)}} L] \tag{A18a}$$

Ignoring spatial dependence of all statistics, we have

$$\boxed{\frac{\mathbb{VAR}[\partial_{x^{(l)}} L]}{\mathbb{VAR}[\partial_{x^{(l+1)}} L]} \approx \left(\frac{\xi^{(l)}}{\rho^{(l)}}\right)^2 \cdot \frac{n^{(l+1)}}{n^{(l)}}} \tag{A19}$$

A2 Experiment Settings

A2.1 General Experiment Settings

Our method is evaluated using several ResNet models on CIFAR. Weights and inputs are quantized to 4-bit, and b_{PIM} varies from 3 to 10. The first convolution layer and the final fully-connection layer for classification are implemented on digital system, namely $b_{\text{PIM}} = +\infty$ for these two layers. To accurately evaluate the inference accuracy on actual hardware with variations, non-linearity, and noise, we evaluate the proposed method using physical models of a state-of-the-art SRAM PIM chip prototype. Each PIM SRAM macro in the chip computes 32 analog MACs ($N <= 144$, b_{PIM}=7) in parallel. The measured 32 transfer functions, shown in Fig. A1, capture all the nonlinearity and mismatch of the physical chip. Random noise in computation, which follows Gaussian distribution and is solely characterized by root-mean-square (RMS) error, is measured to be 0.35 LSB. Due to the small size of the prototype chip, running through all images in test dataset is infeasible in time, so we build a hardware calibrated physical model to quickly, accurately and flexibly simulate the inference accuracy of real hardware, which is a widely adopted common practice in custom hardware research because of the inhibiting costs of building a full-scale chip for large DNNs (Yue et al., 2020; Su et al., 2020; Si et al., 2020; Jia et al., 2020; Jiang et al., 2019). We have experimentally confirmed the identical MAC and inference results of the model and a real physical chip. It is also worth noting that the non-idealities presented in this SRAM PIM chip is representative of that of various types of PIM hardware.

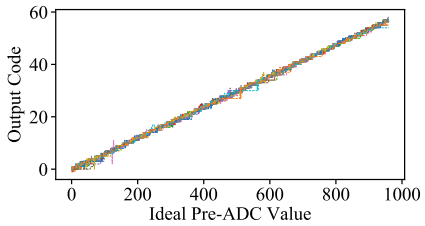

Figure A1: Imperfect MAC outputs from a PIM chip.

To verify the effectiveness of our method, we experiment on ResNet20, ResNet32, ResNet44 and ResNet56 on CIFAR10 and ResNet20 on CIFAR100. Following previous practice (Jin et al., 2020), weights and inputs of convolution and fully-connected layers are quantized to 4-bit ($b_w = b_a = 4$), including the first and last layers, except that inputs to the first layer are kept at 8 bit, and we do not apply normalization on these images. Batch normalization layers and bias in the final fully-connected layers are full-precision. The quantization resolution for PIM system (b_{PIM}) varies from 3 to 10. The first convolution layer and the final fully-connection layer for classification are implemented on digital system, namely $b_{\text{PIM}} = +\infty$ for these two layers. For CIFAR10 and CIFAR100, the 1×1 convolution layers for residual connection require much less computations and thus are also implemented on digital system. For differential and bit serial scheme, inputs are first split along the channel dimension into sub-tensors, each with a unit channel of 16, corresponding to $N = 144$ for 3×3 convolution, and processed separately before summing the final PIM outputs. For native scheme, we instead use a unit channel of 1 and thus $N = 9$ to match the experiment setting in Rekhi et al. (2019) which use $N = 8$. For real-curve results, since we totally have 32 curves (ADC components), each for 8 outputs with $b_w = 4$ bits, the output channels are split with unit output channel of 8.

The input image is randomly cropped to 32×32 and randomly flipped horizontally during training and directly applied without augmentation for inference. All models are trained from scratch with 200 epochs and multi-step learning rate scheduler, where the initial learning rate is 0.1 and reduced by 10 times at the 100-th and 150-th epochs. We use SGD optimizer with Nesterov momentum of weight 0.9 and weight decay is set to 0.0001. Batch size is 128 for all experiments. We apply constant rescaling on all layers, including the convolution layers, in contrast to only the last fully-connected layers

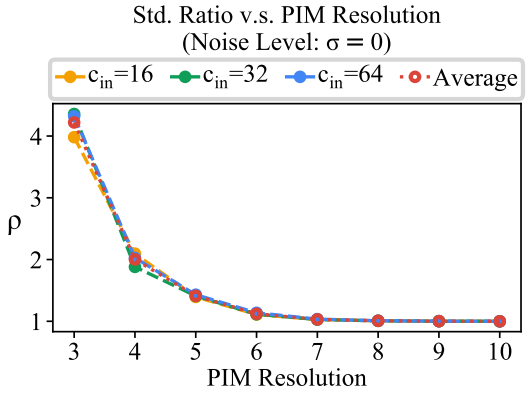

Figure A2: Impact of PIM quantization on the output scale measured by standard deviation. ρ is defined as in (5d).

suggested in (Jin et al., 2019), despite batch normalization is applied in the model. All experiments are finished on one GeForce GTX 1080 GPU with 12GB memory.

Weights are quantized with a modified DoReFa scheme, without mapping between intervals of $[-1, 1]$ and $[0, 1]$. Specifically, the quantized weights are given as

$$Q_i = \frac{1}{2^{b_w-1}-1} s \cdot \text{round}\Big((2^{b_w-1}-1) \cdot \frac{\tanh(W_i)}{\max_k |\tanh(W_k)|}\Big) \tag{A20a}$$

$$s = \frac{1}{\sqrt{n_{\text{out}} \mathbb{VAR}[Q_i]}} \tag{A20b}$$

where n_{out} denotes the number of output neurons of the linear layer. For native scheme, since the output can also be negative, we also adopt such quantization function.

A2.2 Error Analysis Experiment Settings

Computing Error Analysis (Fig. 3) For this example, we first obtain the MAC results via uniform random sampling on the output space, and apply PIM quantization together with noise injection. By comparing the ideal output with the noisy quantized output for different noise levels, we can obtain the errors, from which we estimate the standard deviation of them, for each value of noise levels. These standard deviations are then normalized by that for the noiseless quantization.

A3 Scale-Enlarging Effect of PIM Quantization

Here we show the scale-enlarging effect of PIM quantization. Specifically, we study an idealized noiseless system to examine the effect of **Q** on the standard deviation of outputs. As an example, we experiment on a toy example of convolution with bit serial scheme, and calculate standard deviation ratio between the outputs with and without PIM quantization. For this purpose, we set the number of input channels to 16 and that of output channels to 32. The kernel size is given by 3×3, and both inputs and weights are quantized to 4 bit. We experiment on a random batch of 100 data, each distributed uniformly on $[0, 1]$ before quantized. Weights are randomly sampled with normal distribution under Kaiming intialization condition (He et al., 2015), and quantized with the previously mentioned modified DoReFa scheme (Zhou et al., 2016), given by (A20a). We experiment on an ideal bit-serial PIM system, and calculate standard deviation ratio between the output of PIM system and that from conventional quantization with digital accelerator. We plot this ratio against PIM bit-width (b_{PIM}), and obtain such curves for different numbers of input channels. As illustrated in Figure A2, we find that the difference between the two scenarios is not significant for high precision, which is as expected. However, for mediate precision such as $5 \sim 7$ bits, they start to become different, and for ultra-low bit-widths, such as $3 \sim 4$ bits, the discrepancy can be as large as $2 \sim 4$.

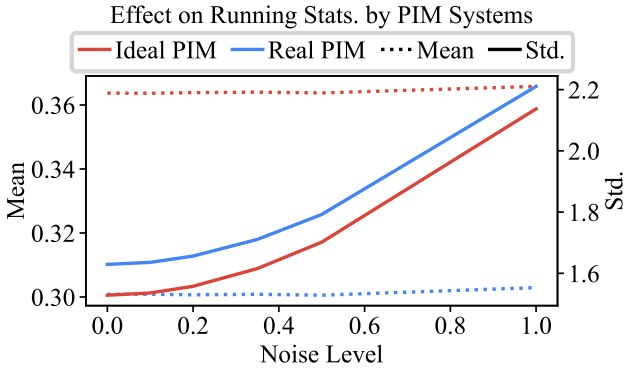

Figure A3: Impact of nonlinearity and noise non-idealities on the running statistics. Note this is one sampling results.

Table A1: Scaling factor for forward rescaling for different PIM resolution b_{PIM} and different PIM decomposition schemes.

b_{PIM}	Native	Differential	Bit Serial
3	100	1000	100
4	20	1000	30
5	1	1000	30
6	1	1000	30
7	1	1000	1.03

A4 Impact of Non-idealities on BN Statistics

In this section, we study the impact of non-ideal effects on BN statistics. We experiment with a toy example of one layer convolution implemented on ideal or real PIM systems, and calculate the running statistics of output for different noise levels. For this purpose, we use the same toy experiment setting as in A3, and set the unit output channel for real-curve inference to 8. The results are illustrated in Figure A3. We find that output statistics can change by as much as 30%, which might have significant impact on the model's final output, especially if its behavior is sensitive to these values.

A5 Scaling Factors for Forward Rescaling

Here we list the rescaling factors for forward propagation, as shown in Table A1. We find that it depends on PIM resolution b_{PIM} and PIM decomposition scheme. Moreover, it can even be different for different software package versions. As mentioned in the text, the underlying reason is still unclear to us.

A6 Ablation Study

Here we provide more in-depth ablation study for our methods, including of PIM quantization-aware training, the rescaling techniques, and batch normalization calibration.

A6.1 PIM-QAT and Rescaling

We first study the effectiveness of PIM-QAT together with the rescaling techniques. To this end, we compare the baseline with that trained with PIM-QAT, without using BN calibration or adjusted bit training. The two rescaling techniques for forward and backward propagation are applied, and we use

noiseless ideal PIM system, without using any real chip curve. Table A2 compares the results of bit serial scheme for several different b_{PIM}'s, which are also plotted in Figure A4. It can be seen that for low b_{PIM}, our method is significantly better than the baseline. Specifically, for $b_{\text{PIM}} < 9$, our method gives better results, and for ultra-low bit-width, such as 3 bit, where baseline models are not different from random guess, our method can still get a reasonable top-1 accuracy of 61.8%. We also find that for sufficient high b_{PIM} (larger than 8), baseline can be better. This is also reasonable as the noiseless PIM with such high precision will almost introduce no precision loss.

Table A2: Accuracy of ResNet20 on CIFAR10 with idealized bit-serial PIM-quantization, where noise or real chip curve are not involved. For our results, we use rescaling techniques for both forward and backward propagation, as described in the text.

b_{PIM}	Method	Acc.	b_{PIM}	Method	Acc.
3	Baseline	10.0	7	Baseline	85.8
	Ours	**61.8**		**Ours**	**90.8**
4	Baseline	10.2	8	Baseline	90.3
	Ours	**77.2**		**Ours**	**90.8**
5	Baseline	11.0	9	Baseline	**91.2**
	Ours	**86.5**		Ours	90.8
6	Baseline	41.1	10	**Baseline**	**91.6**
	Ours	**89.5**		Ours	90.8

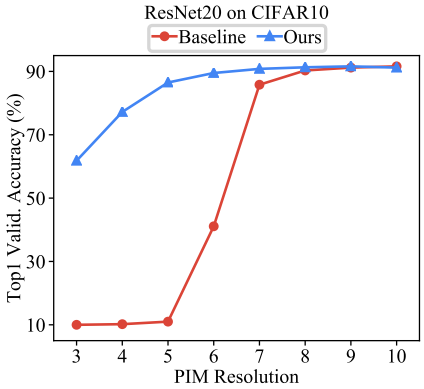

Figure A4: Comparing our method of PIM-quantization-aware training with baseline on idealized noiseless bit-serial PIM systems with different resolutions.

A6.2 RESCALING

We then study the rescaling techniques we propose for both forward and backward propagations. As listed in Table A3 and shown in Fig. A5, for bit serial scheme, if the b_{PIM} is lower than 6, training without forward or backward propagation will all make the training unstable. These experiments demonstrate that both rescaling techniques we propose are necessary and beneficial for stablized training dynamics of the neural network. Experiments on native and differential schemes give similar results.

A6.3 BN CALIBRATION

Besides training techniques discussed above, the discrepancy between training with idealized quantization and inference with real-case non-idealities, including non-linearity and noise, are dealt with BN calibration. To verify its effectiveness, we compare the results using the BN calibration or not for both baseline and our method, and illustrate the results for 7 bit ideal and real PIM in Figure A6. We find that BN calibration significantly improves the results for all cases, especially for those with

Table A3: Ablation study of forward and backward rescaling techniques for bit-serial PIM systems with different resolutions. The accuracy results are based on ResNet20 on CIFAR10.

b_{PIM}	Rescaling Forward	Rescaling Backward	Acc.
3	N	N	10.0
	N	Y	17.1
	Y	Y	61.8
4	N	N	61.0
	N	Y	76.7
	Y	Y	77.2
5	N	N	10.3
	N	Y	17.5
	Y	Y	86.5
6	N	N	10.3
	N	Y	89.1
	Y	Y	89.5
7	N	N	88.8
	N	Y	91.0
	Y	Y	90.8

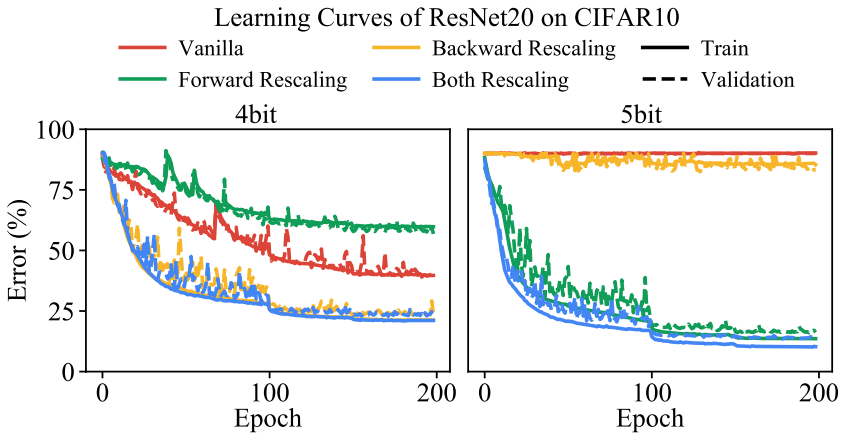

Figure A5: Learning curve comparison with ResNet20 on CIFAR10 for bit-serial PIM system.

real PIM system. More interestingly, it also improves the baseline results, yet the performance is still unsatisfactory and significantly worse than ours. These experiments demonstrate that the change of BN running statistics caused by nonlinearity and noise effects of real PIM systems has strong impact on the predictive capability of the neural network, and this can be alleviated to a large extent with a simple yet effective software solution, without extra training efforts.

A7 More Study of BN Calibration

Here we present more study of the effectiveness of BN calibration, and demonstrate that it is also beneficial for hardware calibrating. Specifically, we use several PIM quantization transfer curves with variation in gain and offset, as illustrated in Fig. A7. The gain and offset variation is extracted from a real chip before hardware calibration. As shown in Table A4, directly applying these curves on a pretrained model leads to random guess results, while BN calibration is able to repair the model and recover the result to reasonable final accuracy.

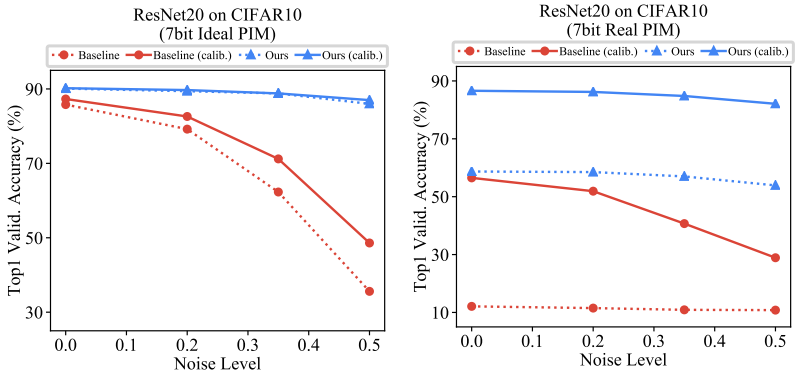

Figure A6: Effect of BN calibration for bit-serial PIM systems with idealized and real curve quantization. We can find BN calibration helps for both our method and the baseline.

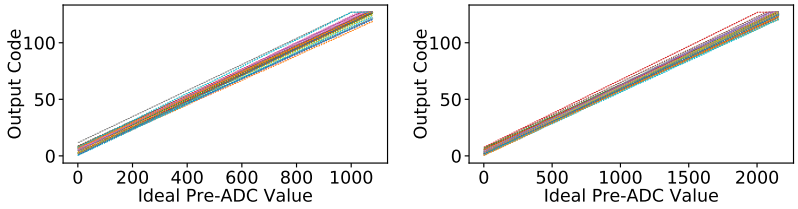

Figure A7: Generated idealized 7bit curves with gain and offset variations, where $N = 72$ for (a) and $N = 144$ for (b). Gains and offsets are both sampled with normal random distributions, where $\mathcal{N}_{\text{offset}} \sim (0, 2.04)$ and $\mathcal{N}_{\text{gain}} \sim (1, 0.024)$. The standard deviations for them are determined based on real-chip testing results.

Table A4: Accuracy of ResNet20 and ResNet56 on CIFAR10 with idealized bit-serial PIM-quantization with gain and offset variations, where noise or real chip curve are not involved.

Depth	N	Gain & Offset Variation	BN Calib.	Acc.
20	72	N	-	91.2
		Y	N	10.0
		Y	Y	**90.7**
	144	N	-	90.8
		Y	N	10.0
		Y	Y	**90.6**
56	72	N	-	92.2
		Y	N	10.0
		Y	Y	**91.7**
	144	N	-	90.3
		Y	N	10.1
		Y	Y	**89.7**