Device Variation Effects on Neural Network Inference Accuracy in Analog In-Memory Computing Systems

*Qiwen Wang, Yongmo Park, and Wei D. Lu**

Qiwen Wang, Yongmo Park, Prof. W. D. Lu
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109, U.S.
E-mail: wluee@umich.edu

Keywords: Deep neural network, RRAM, emerging memory, In-Memory Computing, Process-In-Memory, Analog Computing

In analog in-memory computing systems based on non-volatile memories such as resistive random-access memory (RRAM), neural network models are often trained offline and then the weights are programmed onto memory devices as conductance values. The programmed weight values inevitably deviate from the target values during the programming process. This effect can be pronounced for emerging memories such as RRAM, PcRAM, and MRAM due to the stochastic nature during programming. Unlike noise, these weight deviations do not change during inference. We investigate the performance of neural network models against this programming variation under realistic system limitations, including limited device on/off ratios, memory array size, ADC characteristics, and signed weight representations. We also evaluate approaches to mitigate such device and circuit non-idealities through architecture-aware training. The effectiveness of variation injection during training to improve the inference robustness, as well as the effects of different neural network training parameters such as learning rate schedule, will be discussed.

1. Introduction

Deep neural networks (DNNs) have achieved unprecedented capabilities in tasks such as image and voice analysis and recognition and have been widely adopted. However, computation requirements and the associated energy consumption of neural network implementations have been growing rapidly[1]. In addition, traditional computing architectures are ineffective for DNN workloads due to the high memory access demands, making it even more challenging to meet these computational requirements. Many systems based on digital CMOS technology have been developed specifically for accelerating DNN workloads, including GPU, FPGA, and more specialized accelerators like DPUs. While these systems have shown significant improvements over traditional CPUs in both computing power and energy efficiency, continued innovation is necessary to meet the growing demand. Particularly, DNN inference workload on edge computing platforms like mobile and IoT has stringent energy efficiency requirements due to limited energy supply, and unconventional approaches like analog computing may prove more advantageous in meeting this requirement.

The most important limiting factor for DNN computing is the transfer of data between processors and off-chip memories due to the limited density of existing on-chip memory technology. IMC systems, utilizing the density advantage of emerging memory technologies like RRAM, can potentially store entire DNN models on-chip, thus eliminating off-chip memory access. Analog IMC systems that utilize the device conductance to directly perform vector-matrix multiplication (VMM) operations further allow device-level parallelism that leads to higher performance[2,3]. Meanwhile, neural networks are known for their fault tolerance, making it a feasible workload for analog computing, which is generally unsuitable for traditional arithmetic operations due to its inherently lower precision. Thus analog IMC systems promise drastic improvement in performance and energy efficiency for DNN applications and have gained much popularity in recent years[4–9]. However, non-idealities in memory devices and peripheral circuits can still cause significant degradation of neural network inference accuracy. In general, for analog computing systems, inference accuracy needs to be ensured before any benefit in energy efficiency can become material.

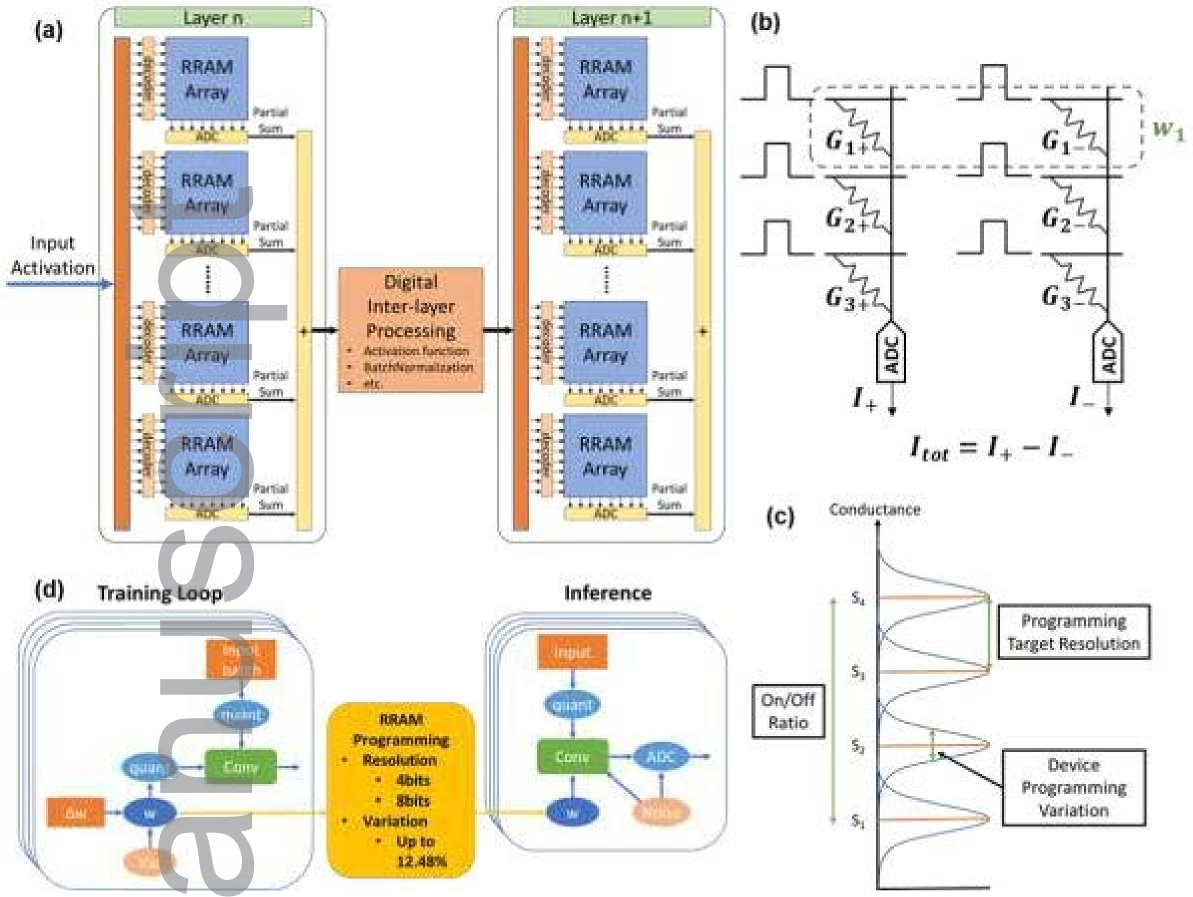


Figure 1. a) Tiled analog in-memory computing systems. Large DNN layers are mapped onto multiple memory arrays. Analog outputs of each array are digitized by ADCs to produce partial sums. The partial sums are then summed in the digital domine to produce the final layer output. b) Signed weights are represented on two memory cells in two different columns. c) device characteristics consider in this study. d) Neural network models are trained off-line then programmed onto memory arrays for inference.

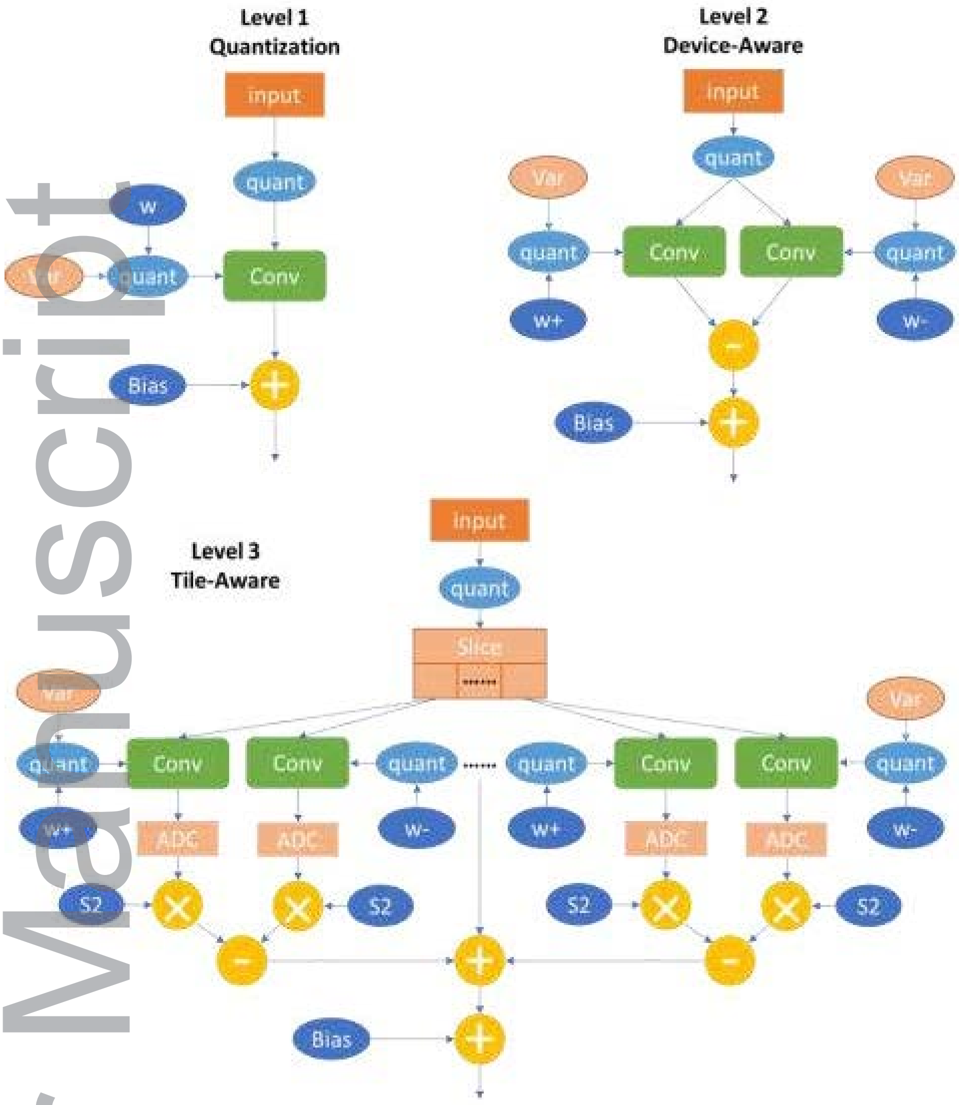


Figure 2. Architecture-aware training topology. We propose that architecture aware training can be considered in 3 levels. Level 1 is the standard quantization-aware training method[10], where high precision weights are passed through a fake-quantization function before computation. At Level 2 device-aware training, signed weight representation in memory cells, and limited on/off ratio are considered. At Level 3 tile-aware training, the limited memory array size and ADC precision limitation are also considered. In addition, for all 3 levels, variation can be injected on per mini-batch basis to mimic the effect of programming variation.

2. Tiled Analog In-Memory Computing System and Architecture-Aware Training

2.1. The Necessity of the Tiled-Architecture

There are 3 types of important non-idealities in analog IMC systems for VMM operations, interconnect parasitics, ADC limitations, and memory device non-idealities. Because energy efficiency is the most important target, ADC operating frequency is likely to be limited to below ~100MHz[11]. At this speed, with more than 10ns of hold time between input change and ADC sampling, in conjunction with the limited array size, transient effects are generally negligible in memory arrays. Therefore, line resistance is the primary parasitic effect. In a large array, the effect of line resistance is dependent on data pattern from all cells in the array and the input signals, and thus can only be compensated by performing expensive calculations based on the memory states and input signals, which defeats the purpose of IMC[12]. To address this issue, the array size must be limited to avoid the effect of line resistance, and large-scale neural networks have to be mapped onto multiple arrays[13]. This is one of the reasons for the tiled architecture analyzed in this study. In our proposed system, the array size is 256 x 64, and assuming line resistance of ~2 Ω per cell, device LRS resistance of 33 kΩ, line resistance will have a negligible effect on the array operation.

2.2. Cell Defects

In this study, we did not consider stuck at fault cell defects. The defects have been considered by many prior studies, including some of our works [14]. Generally, a small portion of stuck at open devices will not have meaningful impacts. However, a shorted device would saturate the output of an entire column. Columns with shorted cells have to be disabled and replaced with spare ones to deal with shorted devices. Although replacing columns means extra areas are needed for spare columns, once the defective ones are replaced, they will not have an impact on the inference accuracy.

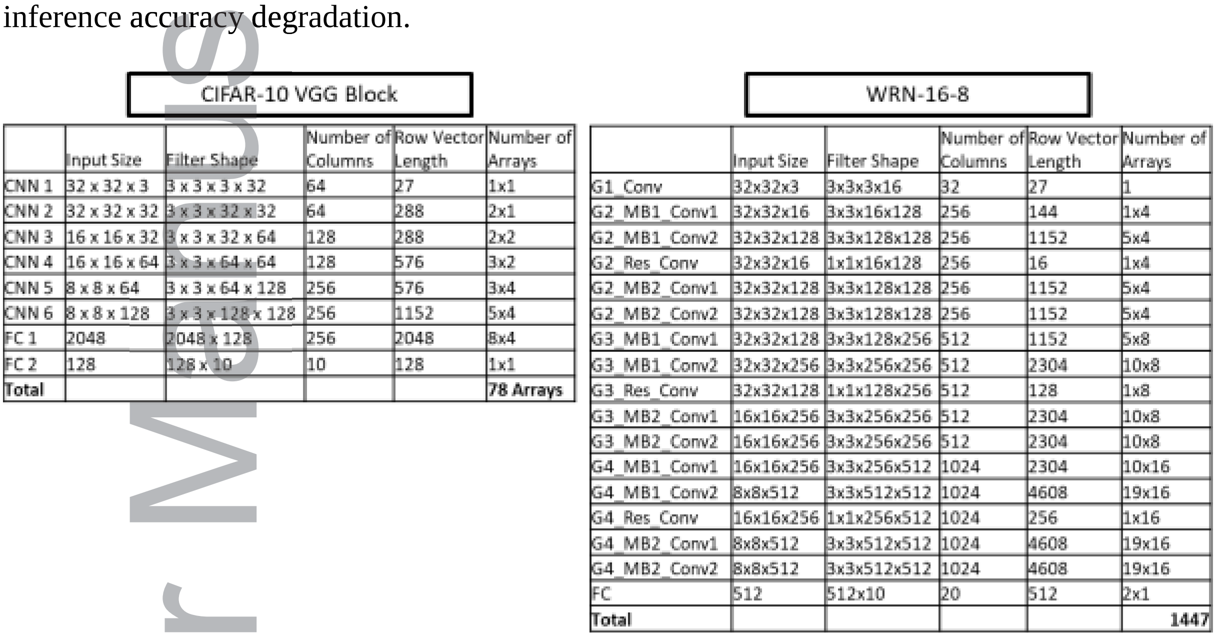
2.3. Error Caused by the Tiled-Architecture

Although the tiled architecture avoids the line resistance effects, the additional computing error caused by this implementation needs to be analyzed, including the effects of ADC limitations and device non-idealities. Several prior studies have been published that discuss approaches to implement large DNN models on practical RRAM arrays using a tiled architecture, as shown in Figure 1[13,15,16]. The effects of limited RRAM array size, ADC precision, signed weights representation in two RRAM cells, and RRAM cell quantization effects in analog IMC systems have been studied. In such a system, neural network models are trained off-line and programmed onto memory arrays, and large neural network layers are mapped onto multiple memory arrays where partial sums (Psums) are produced by ADCs at each array and summed in digital domain[15] (Figure 1a). Signed weights are represented in two cells on two different columns that receive the same input activations. Currents from the two columns are quantized by ADCs individually, then the digital output of the negative column is subtracted from that of the positive column (Figure 1b).

2.4. Architecture-Aware Training

In general, many of the device and circuit non-ideality effects can be effectively mitigated through architecture-aware training methods[15], where hardware details are mimicked in the training process. In architecture-aware training, we developed a simulator based on Google's TensorFlow deep learning framework by modifying the training graph from the standard floating-point pipeline. To compare the impact of different hardware non-idealities, we consider 3 inference pipelines, Level 1 through Level 3, and their corresponding training topologies (Figure 2). In Level 1, only the quantization of weights and activations are considered. In Level 2, the effects of signed weights representation on two cells and limited device on/off ratios are introduced. For both training and inference in Level 1 and Level 2, we used the common scheme for quantization-aware training[10], where the weights pass through the *fakequantization* function before calculations are conducted. The *fakequantization* function does not change the overall range of the weights and instead rounds the weights to a number of fixed values determined by the range and resolution set for the function, and these parameters can be different for each layer. For actual hardware representation of weights where the conductance range is fixed for the whole system, the outputs of each layer need to be multiplied by a high precision scaler to match that of the software model. In Level 3, the physical range of memory cells, the multiplier, limited memory array size, and ADC precision limitation are introduced. As described in Figure 2, separate multipliers are assigned to each array and trained during the training process.

By sequentially introducing different levels of architecture details during the training process, the neural network model can potentially account for these architecture and device factors and recover the desired model accuracy[15]. However, high levels of device programming variation, which is indicative of today‘s analog memory devices, still present challenges in considerable inference accuracy degradation.



CIFAR-10 VGG Block

	Input Size	Filter Shape	Number of Columns	Row Vector Length	Number of Arrays
CNN 1	32 x 32 x 3	3 x 3 x 3 x 32	64	27	1x1
CNN 2	32 x 32 x 32	3 x 3 x 32 x 32	64	288	2x1
CNN 3	16 x 16 x 32	3 x 3 x 32 x 64	128	288	2x2
CNN 4	16 x 16 x 64	3 x 3 x 64 x 64	128	576	3x2
CNN 5	8 x 8 x 64	3 x 3 x 64 x 128	256	576	3x4
CNN 6	8 x 8 x 128	3 x 3 x 128 x 128	256	1152	5x4
FC 1	2048	2048 x 128	256	2048	8x4
FC 2	128	128 x 10	10	128	1x1
Total					**78 Arrays**

WRN-16-8

	Input Size	Filter Shape	Number of Columns	Row Vector Length	Number of Arrays
G1_Conv	32x32x3	3x3x3x16	32	27	1
G2_MB1_Conv1	32x32x16	3x3x16x128	256	144	1x4
G2_MB1_Conv2	32x32x128	3x3x128x128	256	1152	5x4
G2_Res_Conv	32x32x16	1x1x16x128	256	16	1x4
G2_MB2_Conv1	32x32x128	3x3x128x128	256	1152	5x4
G2_MB2_Conv2	32x32x128	3x3x128x128	256	1152	5x4
G3_MB1_Conv1	32x32x128	3x3x128x256	512	1152	5x8
G3_MB1_Conv2	32x32x256	3x3x256x256	512	2304	10x8
G3_Res_Conv	32x32x128	1x1x128x256	512	128	1x8
G3_MB2_Conv1	16x16x256	3x3x256x256	512	2304	10x8
G3_MB2_Conv2	16x16x256	3x3x256x256	512	2304	10x8
G4_MB1_Conv1	16x16x256	3x3x256x512	1024	2304	10x16
G4_MB1_Conv2	8x8x512	3x3x512x512	1024	4608	19x16
G4_Res_Conv	16x16x256	1x1x256x512	1024	256	1x16
G4_MB2_Conv1	8x8x512	3x3x512x512	1024	4608	19x16
G4_MB2_Conv2	8x8x512	3x3x512x512	1024	4608	19x16
FC	512	512x10	20	512	2x1
Total					**1447**

Table 1. Models used for benchmarking. Only CNN and fully connected layers are shown. RRAM array size of 256x64 is used.

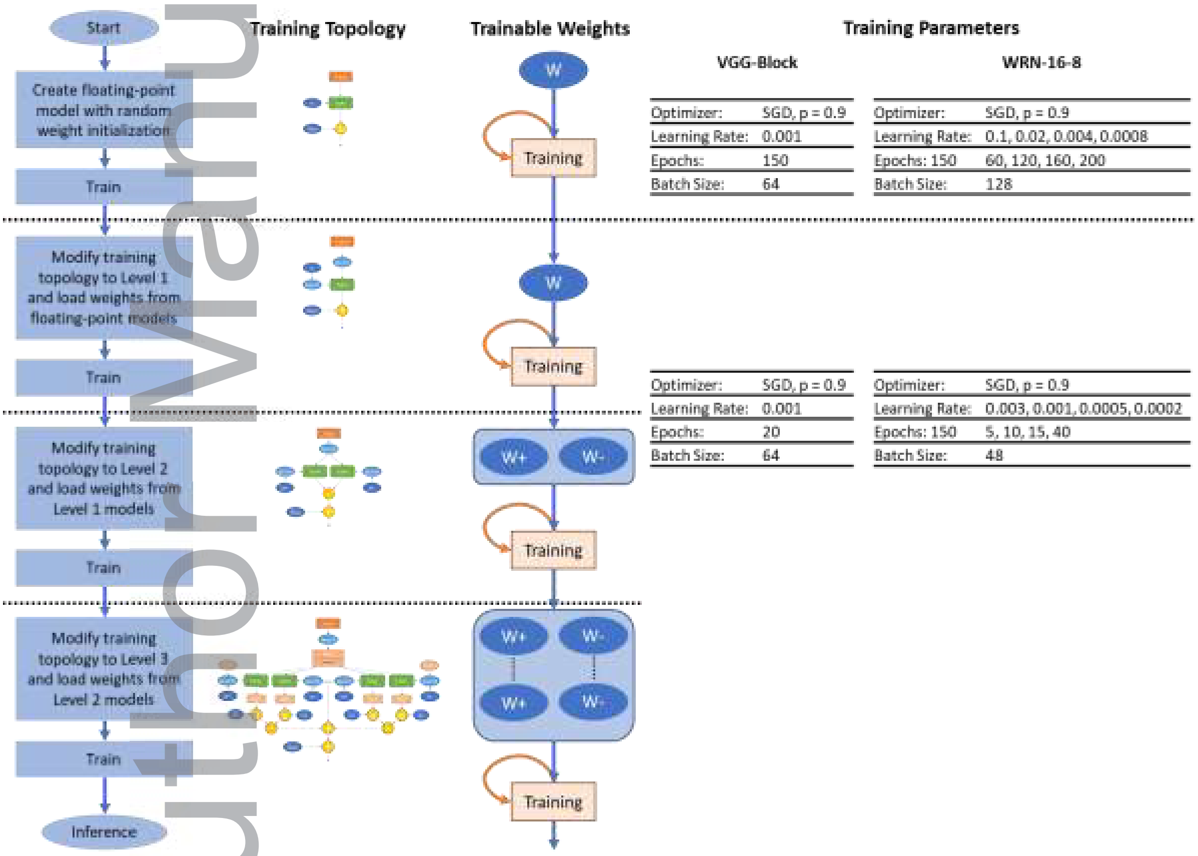
3. System Setup

3.1. Networks Used for Benchmarking

In this study, we chose 3 neural network and dataset combinations of various complexities to investigate the impact of analog IMC accuracy at realistic device non-idealities for different network and dataset complexity (Table 1). The first network is a relatively simple VGG-block-based model trained for the CIFAR-10 dataset. This model contains only convolution (Conv) layers, a fully connected (FC) layer, and MaxPool layers. The second network is the Wide ResNet 16-8 model (WRN)[17]. This network uses residual connections and batch normalization in addition to convolution and fully connected layers. We used the WRN 16-8 network for the CIFAR-10 dataset and the more complex CIFAR-100 dataset to test the effects on more challenging tasks.

3.2. Hardware Characteristics

We used 8bit ADC in our study because it has been found to offer a good balance between energy efficiency and resolution, as reducing resolution further does not appear to yield a meaningful improvement in energy/sample[18]. RRAM cells with an analog read current range of 0.3μA ~ 3μA, ADC input range of 0 ~ 45μA, array size of 265x64 were considered for the tiled implementation.


Start
Create floating-point model with random weight initialization
Train
Modify training topology to Level 1 and load weights from floating-point models
Train
Modify training topology to Level 2 and load weights from Level 1 models
Train
Modify training topology to Level 3 and load weights from Level 2 models
Train
Inference
Training Topology
Trainable Weights
W
Training
W
Training
W+
W-
Training
W+
W-
W+
W-
Training
Training Parameters
VGG-Block
WRN-16-8

Optimizer:	SGD, p = 0.9
Learning Rate:	0.001
Epochs:	150
Batch Size:	64

Optimizer:	SGD, p = 0.9
Learning Rate:	0.1, 0.02, 0.004, 0.0008
Epochs: 150	60, 120, 160, 200
Batch Size:	128

Optimizer:	SGD, p = 0.9
Learning Rate:	0.001
Epochs:	20
Batch Size:	64

Optimizer:	SGD, p = 0.9
Learning Rate:	0.003, 0.001, 0.0005, 0.0002
Epochs: 150	5, 10, 15, 40
Batch Size:	48

Figure 3. Architecture-aware training process and parameters used during training.

3.3. Training Process

We first obtain floating-point models using standard practice. For the VGG-block-based[17] models, we trained for 150 epochs using the stochastic gradient descent (SGD) optimizer with a learning rate of 0.001, momentum of 0.9. For WRN models, we follow the parameters described in[19]. Then, different levels of hardware details are progressively introduced during training, as schematically shown in Figure 3, along with the parameters used during the training processes. Specifically, Level-1 models are fine-tuned from the floating-point models, Level-2 models are fine-tuned Level-1 models, and Level 3 models are fine-tuned from Level-2 models. We found this approach leads to better model inference accuracy compared with training directly the Level-2 or Level-3 models from scratch with random weights[15]. In fact, we found that Level-3 MNIST models trained from random weights reached only 77.78% accuracy (compared to 99.13% for model fine-tuned from Level-2 and float model) in previous studies, and Level-3 VGG and WRN models produced accuracies of only 10%, which is no more than chance for the CIFAR-10 dataset. In the fine-tuning process, we used a learning rate of 0.001 for the VGG-block-based model and trained for 20 epochs. For the WRN models, we used a learning rate schedule, where the learning rate starts at 0.003, then steps down to 0.001, 0.0005, 0.0002 after 5, 10, 15 epochs and trained for a total of 40 epochs.

4. Effects of Computation Errors in Analog IMC Systems

First, we present the effects of deterministic errors including weight and activation quantization, signed weight representation, limited RRAM array size, ADC precision limitations, and RRAM cell on/off ratios (Figure 4). For the 3 network-dataset combinations we studied, when only quantization (activation and weights quantized to 8bits) and signed weight representation were considered during inference (Level 2), there is minimal accuracy drop from just using the quantization-aware trained models[10] (Level 1). We do note that the activation quantization range in the inference pipelines must correspond to the input range of activation function used during training (ReLu6 etc.), or there is severe degradation in accuracy due to the limited range due to the quantization effects.

However, in the presence of a low device on/off ratio and/or array size and ADC limitations, the quantization-aware trained models cannot produce acceptable accuracies. By introducing the finite on/off ratio properties in the training pipeline, device-aware trained models can successfully mitigate the effect of limited on/off ratio down to 10, along with any effects due to the two-column signed weight representation, as shown in Figure 4. On the other hand, in the presence of array size and ADC limitations, the device-aware training, i.e. Level-2 training pipeline, results in poor accuracy for the more complex models or datasets such as WRN. Acceptable results may be produced by Level-2 training for simpler models such as VGG-blocks due to the use of only Conv and FC layers which are generally more resilient to errors. As a result, tile-aware training (i.e. Level 3 pipeline) must be used for the more complex models or datasets to produce good accuracy, as shown in Figure 4. We believe the more complicated model structure with residue connections and the use of batch normalization layers make the WRN models more sensitive to errors. Particularly, models with batch normalization layers are sensitive to changes in activation distribution, and the quantization of partial sums due to ADC precision and range limitations produce a shift in activation distributions[20].

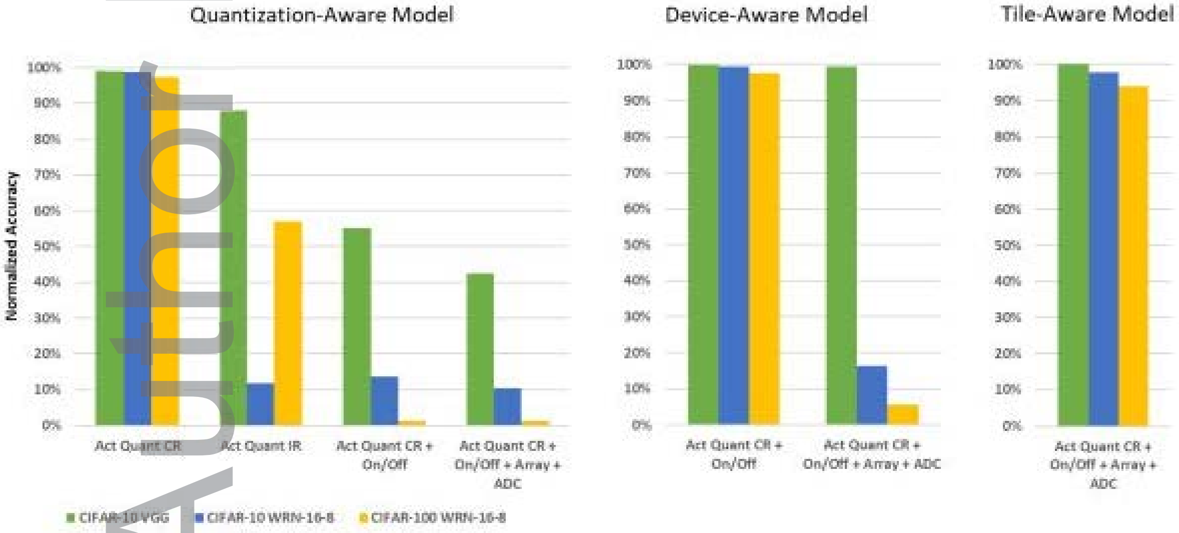


Figure 4. Effect of signed weights represented in two cells, on/off ratio, ADC, and array size limitation. Inference accuracy. Act Quant CR: activation and weight quantization 8bits with activation quantization range corresponding to ReLu6 used during training, on/off ratio 1000. Act Quant IR: activation and weight quantization 8bits with activation quantization range of 0-1 which does not correspond to the ReLu6 range used during training. Act Quant CR + On/Off: low on/off ratio of 10. Act Quant CR + On/Off + Array + ADC: array size 265x64, 8bit ADC. Floating-point accuracies for the CIFAR-10 VGG, CIFAR-10 WRN, CIFAR-100 WRN models are 83.73%, 95.11%, and 74.74%.

5. Programming Variation Effects on Inference Accuracy

Next, we examine the effects of device variations on network inference accuracy. Neural network models are trained off-line then programmed onto memory arrays for inference, and the weights do not change during the inference process. Combined with analog computation, this means any deviations that occur during the device programming process result in inference to be conducted on models that are effectively different from the trained models, leading to potential accuracy degradation. Different from deterministic errors discussed earlier, the randomness of device variations means each programmed chip maps an essentially different model. Re-training each chip individually may potentially recover the accuracy, but will be very expensive and impractical. In the following, we investigate the impact of device programming variation on large-scale DNN networks inference accuracy, the effectiveness of mitigations methods, and factors that impact network robustness against device variation under realistic device and circuit conditions.

We examined the effect of weight variations using models trained with Level 1, 2, and 3 pipelines, and studied the model accuracy in the corresponding inference conditions (e.g. when only quantization effects, quantization + device on/off, and quantization, on/off and finite array size and ADC precision effects are present during inference, respectively) (Figure 5). Previous studies have shown that the VGG-block-based model had minimal accuracy drop even at relatively high variation levels, while more complex models show severe accuracy degradation [15]. In this section, we thus used the more complex WRN-16-8 models for the CIFAR-10 dataset to highlight the effects of device variations.

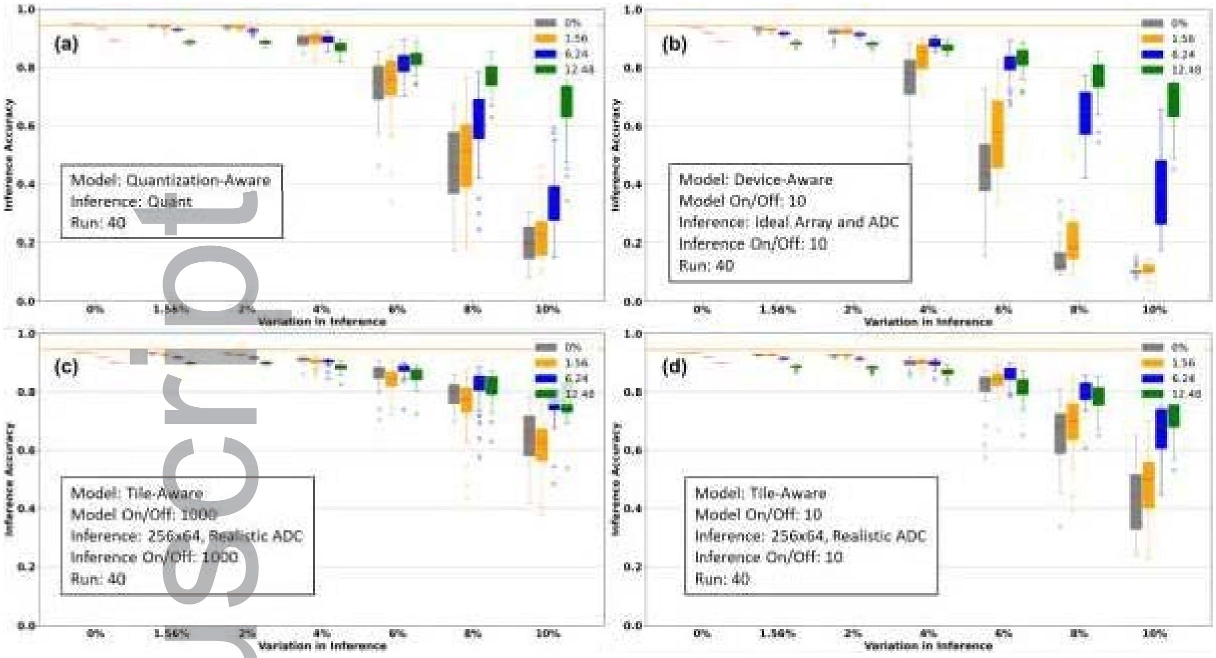


Figure 5. Variation effect under different inference pipelines for WRN-16-8 network on the CIFAR-10 dataset, for models trained with different levels of noise injection. The variation level is defined as standard deviation relative to the dynamic range of the weights. The boxplots show model inference accuracy distribution from 40 runs. Legend: variation injected during training. Orange lines: floating-point baseline. Ideal Array and ADC: no array size limitation, no quantization or range limitation of output. Realistic ADC: 8bit ADC with 0 ~ 45µA as described in section 3.2, array size 256x64.

In the accuracy test, after weight storage, variations were applied additively as Gaussian distributions with a constant standard deviation across all weights (i.e. 4% variation means the standard deviation is 4% of the dynamic range of memory cells). This variation distribution was chosen as a generic example, since memory technologies have substantially different characteristics, and it represents a near-worst-case scenario. On one side, many emerging resistive switching devices exhibit state-dependent programming variation, where lower conductance states are associated with lower variations[5,21], which is less detrimental to inference accuracy. On the other side, programming variations in multi-bit Flash memories are generally more state-independent while also suffering from additional non-linear behaviors[22,23]. In Level-2 and Level-3 inference pipelines, where signed weights are represented in two columns (Figure 1b), the variations are applied independently to each cell. This is different from variations that are directly applied to the signed weights (Level-1) and means the impacts of weight variations are not equivalent between Level-1 and the other pipelines. We also note that the signed weight representation we adopted (Figure 1b) is more realistic than the approach where differential cells (cells consisting of two devices) are read out individually[2,5] and more practical to implement in circuits compared to the approach where 2 rows with positive and negative input voltages signs are used to represent to positive and negative weights.

Because each programming session on each chip results in effectively different models and different inference accuracy, the process was simulated 40 times for each condition. The distribution of inference accuracy is shown in box plots (Figure 5). The models are expected to be programmed onto memory arrays and do not change during inference. Therefore, programming time is less important. Thus, the top 25 percentile in the accuracy distribution is more representative than the average or the median, because it can be achieved by attempting programming sessions multiple times. Gray boxes in Figure 5 represent inference accuracies of models trained without any mitigation measures, and, in general, accuracy degradation becomes unacceptable for variations beyond 4%.

When comparing between Level-1 and Level-2 pipelines, accuracy degradation is more pronounced in Level-2 due to the signed weight representation and low on/off ratio of memory devices (Figure 5a and Figure 5b). When ADC and array size limitation is introduced at Level-3, surprisingly, the accuracy degradation at high variation levels (> 4%) improved compared to Level-1 and Level-2. This is likely due to the presence of trainable S2 multipliers on a per array basis we implemented in the tiled architecture (Figure 2)[15]. At this level, we can also observe the negative effect of a low device on/off ratio (Figure 5c and Figure 5d). However, in general, accuracy degradations become unacceptable when variations exceed 4% of the dynamic range.

As a natural extension in the architecture-aware training approach, we hypothesize that injecting noise during training may improve inference accuracy. Specifically, we used weight noise injection during training to mimic device programming variations to produce trained DNN models that can produce better inference accuracy in presence of variations. In this implementation, weight noise is added after each mini-batch during training, where an error is drawn from a Gaussian distribution for each weight then added to it. The standard deviation for the Gaussian distribution is defined as relative to the dynamic range of the memory cells. For example, 1.56% noise injection means the Gaussian distribution has a standard deviation that is 1.56% of the dynamic range of memory cells. From a general neural network training perspective, noise injection at inputs, hidden units, and weights during training have long been proposed as methods to improve the generalization ability of neural networks[24–28]. In particular, weight noise injection has been shown mathematically to improve fault tolerance as it produces networks with smoother input-output mapping where the output becomes less sensitive to noise[26]. Recent studies have also applied this method to analog computing systems[29–32]. However, these prior studies are generally limited to small-scale networks or did not consider realistic system limitations like ADC characteristics, device on/off ratios, and especially array size limitations. The improvements in inference accuracy from weight noise injection in training can be observed in Figure 5, and the trend in improvements is consistent across different inference pipelines. In general, higher-level noise injection leads to better accuracy recovery. For high device variations, noise injection not only allows the average and the peak accuracy to recover but also reduces the variation in performance between different runs.

The improvements from noise injection can also be observed from model outputs directly. Figure 6a shows the error in model outputs caused by device programming variation with CIFAR-10 validation dataset as input. For inference with a programming variation level of

6.24%, the injection of noises of the same level during training significantly reduces the error in the model outputs. For inference with a higher device programming variation of 12.48%, although substantial errors still occur with 12.48% noise injection, the trend in improvement is similar. When random patterns are used as input for models trained on the CIFAR-10, the error caused by programming variation is significantly larger Figur 5b. This suggests that neural network models are trained for a specific input distribution, and the impact of weight variation can be more pronounced if the inference task is different from that during training.

The robustness of neural network models against weight variation can be characterized as part of the generalization ability. It has been shown that the addition of weight variation exacerbates inference error caused by generalization limitations and roughness of neural network models[26]. This means, in order to obtain acceptable inference accuracy for the same tasks, the models will need to have better generalization ability in the presence of weight variation, and many factors in the training process influence the ability. Indeed, we have found that even when models have similar accuracy with no weight variations, they can have very different robustness against variations. Prior literature has shown ample results for the effects of quantization and learning rate on the generalization ability in standard digital implementation. However, the effects of these factors on neural networks model have not been discussed in the context of analog in-memory computing systems with the presence of realistic hardware limitations. In the next section, we investigate the impact of learning rate, programming target resolution, and different inference pipelines have on model robustness against weight variation.

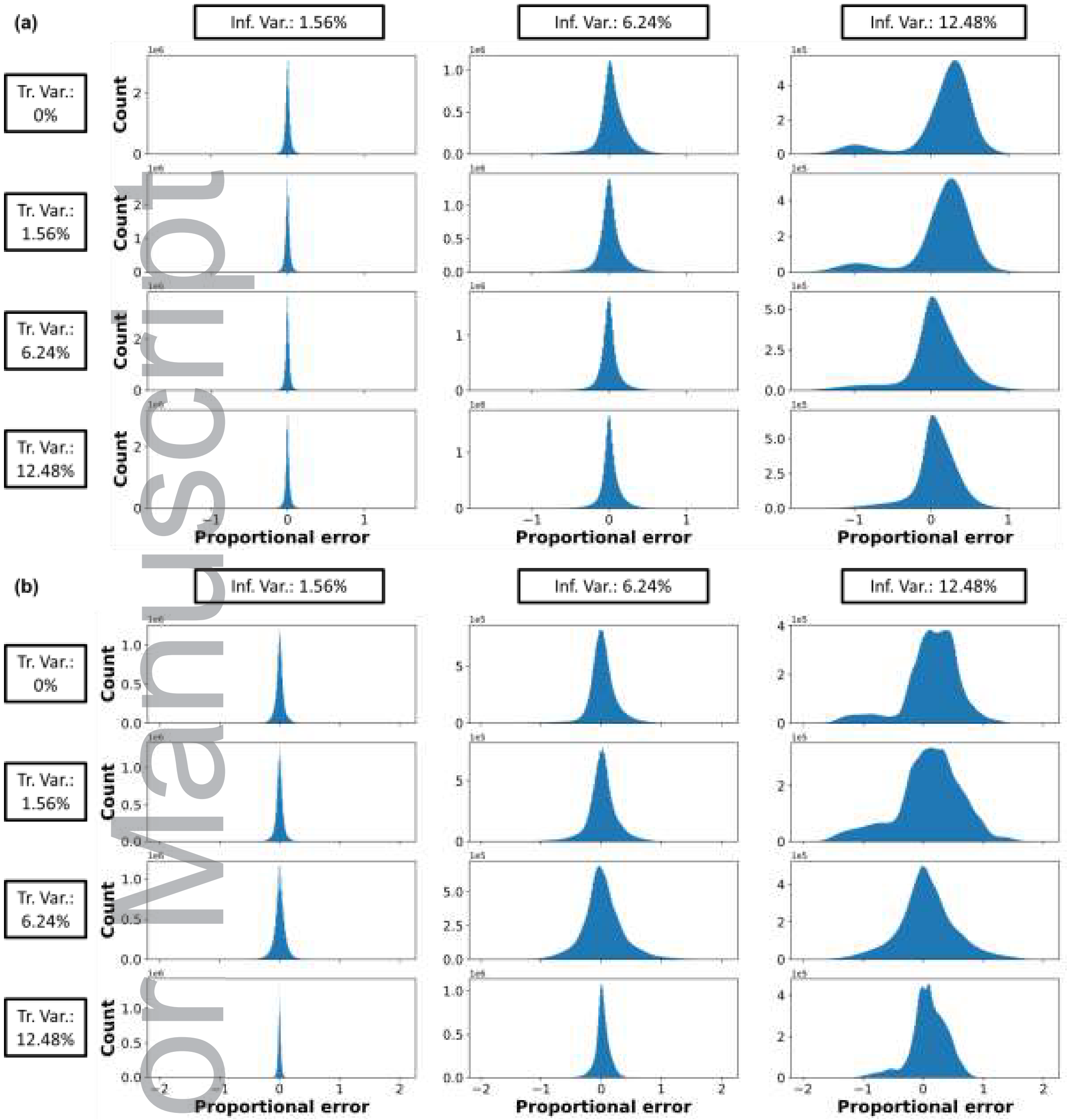


Figure 6. The proportional error of network model inference output with weight programming variation compared to inference output without variation, Level-3 model in Level-3 inference pipeline with on/off ratio of 10. Programming variations are simulated 40 times, and results are aggregated. Tr. Var.: variation injected during training. Inf. Var: variations experienced in the device programming process. (a) images from the validation dataset as input to the network. (b) random pattern as input to the network.

5.1. Effect of Higher Target Programming Resolution

Although 8bit programming target resolution cannot be reliably represented by devices with a variation of even as low as 1.56%, we found, compared to 4bit programming target, higher target resolution produces models more robust to programming variations (Figure 7). Thus, 8bit programming target resolution was used in this work. We believe this is because, although quantization-aware training methods produce models more suited for the specific deterministic error caused by quantization, these models have diminished generalization ability, thus more sensitive to any additional errors like device variation[10,33]. The higher target resolution produces trained models with wider local minima, thus higher robustness against variation in weights.

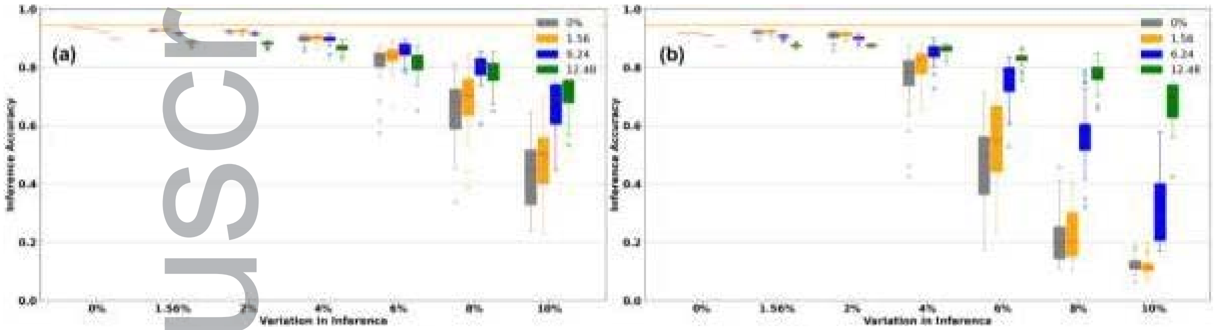


Figure 7. Sensitivity to variation of tile-aware trained WRN-16-8 CIFAR-10 model in tiled inference pipeline with ADC limitation, array size of 265x64, on/off ratio of 10 for both training and inference, accuracy evaluations were run 40 times. a) models trained with 8bit quantized weights and 8bit programming target resolution during inference. b) models trained with 4bit quantized weights and 4bit programming target resolution during inference.

5.2. Effect of Difference Inference Pipeline on the Same Models

We observed that although the same model can generally achieve very similar accuracies under different inference pipelines, in the presence of weight variations very different behaviors are obtained. For example, Level-1 trained models show similar accuracy in the Leve-1 (Figure 5a) and Level-2 (Figure 8a) inference pipelines. However, in the presence of relatively high weight variations, the accuracies are consistently lower in the Level-2 inference pipeline. This is likely because Level-1 trained models are not optimized for the Level-2 inference pipeline when additional hardware details are introduced that are not incorporated during training, but the errors may not be large enough to cause accuracy degradation when there is no weight variation. In the presence of weight variations, the effects are amplified and lead to much worse accuracy degradation when training is not matched with the inference conditions.

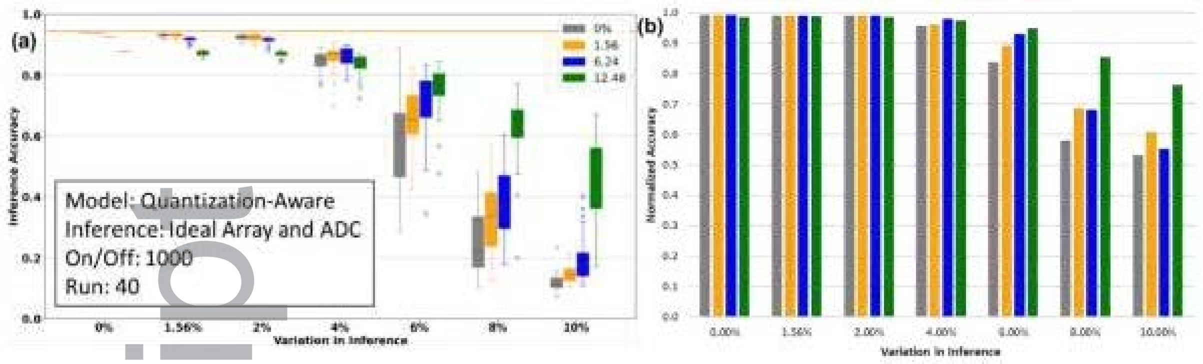


Figure 8.WRN-16-8 models for the CIFAR-10 dataset. a) Accuracy of Level-1 trained model in Level-2 inference pipeline. b) 75th percentile accuracies of Level-1 trained models in Leve-2 inference pipeline, normalized to Level-1 inference pipeline accuracies of coresponding training and inference variation.

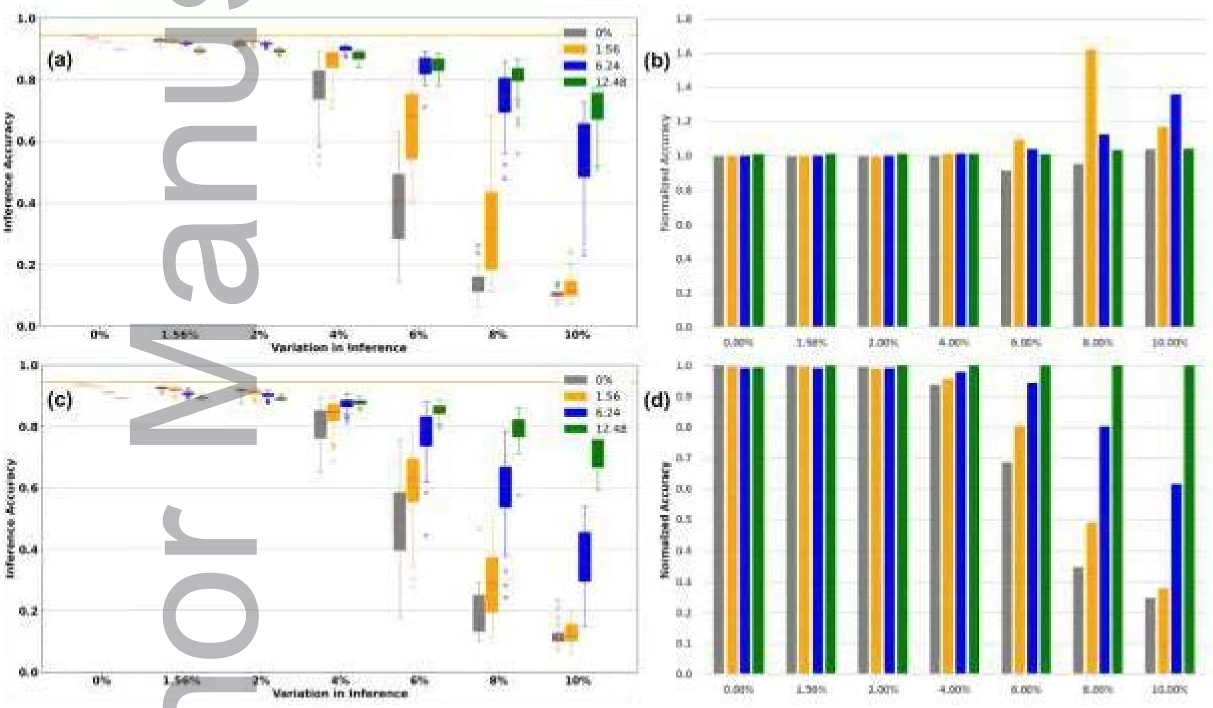


Figure 9. Effects of learning rates. a) Device-aware models trained with a learning rate of 0.0002, evaluated in tiled inference pipeline. b) 75th percentile accuracies in Figure 9a normalized to that of models trained with learning rate schedule (Figure 5b). c) 75th percentile tile-aware models evaluated in the tiled pipeline with an on/off ratio of 10. d) Accuracies in Figure 9c normalized to that of models trained with learning rate schedule (Figure 5d).

5.3. Impact of Learning Rate

When a simple learning rate of 0.0002 is used in the fine-tuning process instead of the learning rate schedule described in Section 2.2, models achieved similar accuracies with no weight variation. When weight variation is introduced, device-aware trained models (Level 2) showed no clear pattern between the two different learning rates, while tile-aware trained

(Level 3) models with a learning rate of 0.0002 are more sensitive to weight variations compared to ones obtained with the learning rate schedule (Figure 9). It is well known that selecting suitable learning rates is critical in the training process for neural network models to converge to optimal states, and learning rate schedules are often superior to constant learning rates[34]. In this particular case, the learning rate schedule showed an advantage in Level-3 training but not in Leve-2 training. We believe the addition of array size and ADC limitations at Level-3, analogous to weight quantization, results in models with less smooth input-mapping, thus illuminating the difference between models produced by the different learning rates.

6. CIFAR-100 Results and Discussions

We also considered the WRN-16-8 model for the more complex CIFAR-100 dataset. The results showed a similar general trend to the results for the CIFAR-10 dataset, while models are much more sensitive to variations across the board with significant accuracy degradation at as low as 2% variation (Figure 10). This, again, illustrate that larger-scale networks, as well as more complex tasks, are more sensitive to weight variations during inference, and as neural networks and tasks become more complicated, further improvements in DNN model robustness and device performance may be required.

Other than improving the intrinsic precision of memory devices, two main methods have been proposed to improve weight storage precision. The first is using multiple cells to encode different bits of one weight[21,35], and the second is to use closed-loop, write-verify programming schemes[36,37]. Using multiple cells drastically decreases memory density and incurs additional peripheral circuit overhead, thus resulting in decreases in computing efficiency in terms of both area and energy. Closed-loop programming processes have already been widely implemented, but have not been able to yield programming precision high enough for 8bit or even 4bit weights. As shown in Figure 10, even 2% variation can lead to significant accuracy degradation for complex tasks.

Here we provide another possibility for future device and programming algorithm design. Neural networks are generally sparse, where weights close to zero constitute a large portion of all weights. This means programming variations at low conductance states have much higher impacts, and non-uniform device programming variation characteristics can be engineered to minimize the effect at low conductance value. In particular, most resistive switching memory technologies have a limited analog dynamic range where conductance can be changed continuously, compared to the entire dynamic range. For weights close to zero, the corresponding memory devices can be hard reset, where conductance is set to the lowest possible value beyond the analog range. This would greatly reduce weight variation for weights close to zero because the absolute variation at the lowest conductance state is generally much lower compared to that inside the analog range. Although hard reset can have an impact on endurance in some memory technology, programming is expected to be infrequent, and endurance is unlikely to be the primary limiting factor.

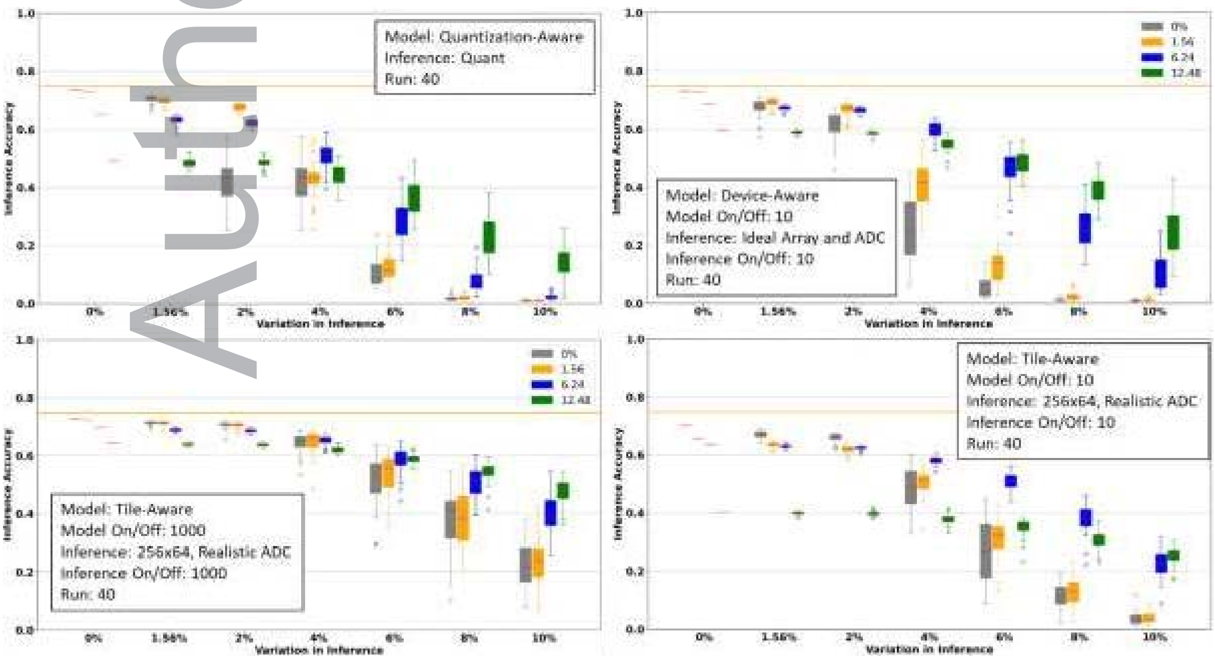


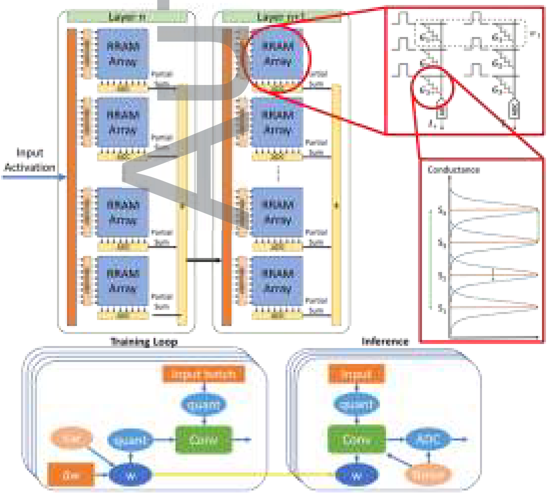
Figure 10. Variation effect under different inference pipeline for WRN-16-8 network on CIFAR-100 dataset.

7. Conclusion

In this work, we took a systematic look at the weight variation effect caused by memory device programming in analog IMC systems, which appears to be the most difficult error source to mitigate. We show proper noise injection can improve model robustness against weight variations. However, in the presence of moderate to high variations and for complex tasks and models, these methods may not be able to fully recover the accuracy drop. Thus, further developments in algorithms to produce neural networks that are more robust against weight variations could be critical for practical deployment for analog IMC systems for neural network workload.

Acknowledgements
This work was supported in part by SRC and DARPA through the Applications Driving Architectures (ADA) Research Center, and by the National Science Foundation through grant CCF-1900675.

Received: ((will be filled in by the editorial staff))
Revised: ((will be filled in by the editorial staff))
Published online: ((will be filled in by the editorial staff))



In non-volatile-memory-based analog in-memory computing systems, variation in the device programming process can cause neural network inference accuracy degradation since the stored weights are different from those in the original model. We investigate the performance of deep neural network models against this programming variation under realistic system limitations, including limited device on/off ratios, memory array size, circuit characteristics, and signed weight representations.