Supplementary Materials for “Robust distillation for compute-in-memory: Realizing reliable intelligence using imperfect memristors”

Wenbin Li[1#], Dongdong Ren[1,2#], Zhiping Wu[1,2#], Dehe Kong[3#], Liuyuan Wen[1], Li Du[4], Yuan Du[2], Yuekun Yang[3], Yinghuan Shi[1], Cong Wang[3], Feng Miao[3], Hongbing Pan[1,2*], Yang Gao[1*]

[1]*State Key Laboratory for Novel Software Technology, Nanjing University, China.*
[2]*School of Electronic Science and Engineering, Nanjing University, China.*
[3]*Institute of Brain-Inspired Intelligence, School of Physics, Nanjing University, China.*
[4]*School of Integrated Circuits, Nanjing University, China.*

Contents

# These authors contributed equally to this work.
* Correspondence to: Hongbing Pan (Email: phb@nju.edu.cn, ORCID: 0000-0002-7181-8278), Yang Gao (Email: gaoy@nju.edu.cn, ORCID: 0000-0002-2488-1813).

1 Supplementary Note

1.1 Supplementary Note 1: Non-idealities in computing-in-memory

Analog Compute-in-Memory (CIM) chips have emerged as a significant development direction in the field of artificial intelligence hardware. By tightly integrating data storage and computation, CIM architectures substantially reduce data movement overhead while greatly enhancing computational efficiency. However, the inherent non-idealities of memory devices pose a severe constraint on model performance in practical deployments [1]. These include process variations, non-linear characteristics, thermal noise, electromagnetic interference, and so on. These variations can lead to inaccuracies in stored weights during model inference and may even significantly degrade the model's inference accuracy. For instance, on RRAM-based CIM platforms, fluctuations in the physical properties of memory cells can cause their conductance to deviate during programming, leading to weight discrepancies [2]. These errors can be amplified as they propagate through the network's layers, ultimately impacting the model's overall inference accuracy. Below, we briefly analyze four typical categories of non-idealities in CIM architectures.

1.1.1 Weight quantization error

When deploying deep neural network models onto CIM chips, weights typically need to be quantized from high-precision floating-point formats to low-precision fixed-point formats, such as 8-bit integers or fewer. This quantization process inevitably introduces errors, known as weight quantization errors. These errors affect the final neural network inference after the weights are deployed on-chip, leading to a degradation in model performance.

1.1.2 Retention loss

Currently, various devices used to implement analog-like CIM chips commonly face the issue of *retention loss*. For example, RRAM and PCM devices exhibit conductance relaxation, while Flash memory shows threshold voltage drift. Retention loss refers to the phenomenon where the value represented by a device shifts shortly after programming or gradually drifts over time, thereby affecting storage precision and model stability.

1.1.3 Thermal drift

PVT (Process, Voltage, Temperature) variations are a widely recognized concern in integrated circuit design, and analog CIM chips are no exception. In particular, thermal drift—the phenomenon where overall chip performance changes under different operating temperatures—can also cause stored weight values to drift, further impacting the accuracy and stability of the model during the inference phase.

1.1.4 Other noise sources

In addition to the aforementioned non-idealities, random noise within the system also cannot be ignored [3]. Such noise primarily originates from the hardware circuitry itself, including noise during the weight write and read processes, noise from analog-to-digital conversion circuits, and interference introduced by input driver circuits. These noise sources are typically stochastic in nature and difficult to predict directly. Therefore, they require statistical analysis based on extensive experimental data and the construction of corresponding noise models to achieve high-fidelity system simulation.

Compensation strategies for non-idealities in CIM architectures can be broadly categorized into two main approaches. One approach is at the software level, which involves incorporating non-ideal factors into the neural network training process through accurate chip modeling and simulation, and introducing corresponding robustness constraints into the network architecture to enhance the model's adaptability to device perturbations. The other approach is at the hardware level, which involves introducing auxiliary hardware modules to actively compensate for non-ideal errors, but this typically incurs additional area costs and power consumption overhead. Therefore, how to effectively mitigate non-idealities in CIM architectures to ensure system stability and energy efficiency has become a key research topic of wide concern in both academia and industry. At the software level, methods such as variation-aware training and ensemble learning have been proposed to address the impact of device non-idealities in CIM chips. These techniques aim to enhance the model's tolerance to hardware noise and perturbations, offering effective technical pathways to solve the aforementioned challenges.

1.2 Supplementary Note 2: Variation-aware training for CIM

Variation-aware training is an algorithmic approach designed to mitigate the non-idealities inherent in Computing-in-Memory (CIM) devices. Its core principle involves integrating hardware variations directly into the neural network training process. By optimizing the model with simulated or statistical representations of these variations, it develops greater resilience to device non-idealities. Recent studies on variation-aware training have proposed the following approaches. Liu et al. [4] mitigated the impact of hardware variations by adjusting weights during training and incorporating the non-ideal errors into the loss function. Other methods [5, 6, 7] train neural networks by directly modeling non-idealities as functions of random variables. Zhu et al. [5] represent device variations and noise as correlated random variables, incorporating global error compensation to optimize the loss and ensure stable inference accuracy. Long et al. [6] improved network robustness to parameter variations by injecting random noise into the parameters during the model training process. Chen et al. [7] proposed an accelerator-aware neural network training method that determines the optimal mapping between weights and memristors using a weighted bipartite matching algorithm. This ensures that weights with a significant impact are not mapped to defective memristors.

Some approaches also introduce specialized constraints during gradient computation and backpropagation to adapt to the computational characteristics of CIM hardware. For instance, to address the asymmetric write characteristics of memristors, researchers have designed weighted gradient update strategies. These strategies mitigate hardware update inconsistencies by accounting for directional differences in memristor writes during weight updates. Similarly, for non-linear memory cells, variation-aware training optimizes model adaptability by either constraining the weight value range or introducing non-linear compensation functions. Moreover, several studies integrate variation-aware training with meta-learning and transfer learning to further improve model robustness to hardware variations. By training an initial model that can rapidly adapt to hardware changes, variation-aware training can achieve fast transferability across different hardware platforms and application scenarios.

In summary, variation-aware training offers an effective algorithmic approach to enhance model accuracy by explicitly modeling and optimizing for the non-idealities of CIM hardware. However, these methods rely on prior knowledge of hardware variation characteristics, which requires testing and statistical analysis for each manufactured chip. Otherwise, a mismatch can occur between the simulated variations during training and the actual variations in the deployed device, preventing the attainment of optimal performance.

1.3 Supplementary Note 3: Ensemble learning for CIM

Ensemble learning is a machine learning technique that combines multiple base learners to enhance overall performance. Recently, ensemble learning has played a significant role in addressing the non-idealities of CIM chips. The non-idealities of CIM devices typically lead to performance degradation in a single model. By combining the outputs of multiple models, ensemble learning mitigates errors from individual models and enhances overall system robustness. In the context of CIM architecture optimization, ensemble learning methods are primarily applied in two forms [8, 9]. The first involves hardware-level integration via multi-copy redundancy. For example, multiple copies of the same model can be mapped to different memory arrays, and their outputs are combined through a majority vote to make the final decision. This approach can significantly improve the system's noise tolerance but at the cost of increased hardware resource overhead. The second is through software-level model fusion.

This strategy promotes model diversity during training by generating multiple sub-models with different initializations or by applying random weight perturbations. The predictions of these sub-models are then fused through methods like weighted averaging or stacking. Thus, aggregating predictions from multiple models improves tolerance to non-ideal errors in CIM systems. Although ensemble learning significantly improves performance, it also introduces drawbacks, including high computational cost, extended training time, and limited interpretability.

1.4 Supplementary Note 4: The proposed multi-teacher knowledge distillation framework

To address the impact of device non-idealities in computing-in-memory chips, we propose leveraging the properties of multi-teacher knowledge distillation to mitigate the loss in model accuracy caused by non-idealities. Knowledge Distillation (KD) is a training paradigm where a smaller student network learns to mimic the behavior of a larger, pre-trained teacher network. This is typically achieved by training the student to match the teacher's output distribution, often using a loss function based on Kullback-Leibler (KL) divergence. To ensure the effectiveness of distillation, some works, such as Bag of Tricks [10], have suggested that the teacher and student networks should share a similar or identical topology.

The concept of KD can be extended to a multi-teacher framework, where students learn from a collective of teacher models instead of a single one. We take advantage of the concept of merging multiple teacher networks proposed by You [11] and Shen et al.[12]. The output of the ensemble teacher model ($\hat{y}^{t_E}$ is typically composed of the average of the predictions ($\hat{y}^{t_i}$ of the individual teacher models. The student model is trained using this average output to minimize the KL divergence:

$$\mathcal{L}_{KD}(\hat{\boldsymbol{y}}^{t_E}, \hat{\boldsymbol{y}}^{s}) = D_{KL}(\hat{\boldsymbol{y}}^{t_E}, \hat{\boldsymbol{y}}^{s}), \quad \text{where} \quad \hat{\boldsymbol{y}}^{t_E} = \frac{1}{T}\sum_{i=1}^{T}\hat{\boldsymbol{y}}^{t_i} \tag{1}$$

The topology of each teacher network does not necessarily belong to the same family as that of the student network. Furthermore, there exists an intrinsic connection between knowledge distillation using the average prediction ($\hat{y}^{t_E}$) from teacher models subjected to different noises and simultaneously learning from multiple teachers under specific conditions:

$$\begin{aligned} D_{KL}\left(\hat{\boldsymbol{y}}^{s}, \frac{1}{T}\sum_{i=1}^{T}\hat{\boldsymbol{y}}^{t_i}\right) &= \sum_{j=1}^{K}\hat{y}_j^s \log\frac{\hat{y}_j^s}{\frac{1}{T}\sum_{i=1}^{T}\hat{y}_j^{t_i}} \\ &\leq \sum_{j=1}^{K}\hat{y}_j^s \log\frac{\hat{y}_j^s}{\left(\prod_{i=1}^{T}\hat{y}_j^{t_i}\right)^{\frac{1}{T}}} \\ &= \frac{1}{T}\sum_{i=1}^{T}D_{KL}(\hat{\boldsymbol{y}}^{s}, \hat{\boldsymbol{y}}^{t_i}) \end{aligned} \tag{2}$$

Here, the subscript j refers to the j-th class. This inequality shows that the distillation loss against the averaged prediction serves as a lower bound for the sum of individual distillation losses from each teacher. However, some studies [13, 14] have shown that simply averaging individual predictions may neglect the diversity and varying importance of the multiple teachers. Therefore, these methods aggregate the outputs of multiple teachers through a gating module, as shown in the following formula:

$$\mathcal{L}_{Ens}^{logits} = H\left(\sum_{i}^{m} g_i N_{t_i}(x), N_s(x)\right) \tag{3}$$

where g_i is the gating parameter used to regulate the contribution of each teacher. Multi-teacher knowledge distillation methods can provide richer and more diverse learning information for the student model. However, the feature representation of each teacher at a specific layer is different. Therefore, how to effectively fuse these representations becomes a key problem. Park et al. [15] proposed feeding the student's feature maps into non-linear layers and then training the output layer to simulate the final feature map of the teacher network. In contrast, Liu et al. [16] proposed having the student model imitate the learnable transformation matrices of the teacher model. Simply averaging predictions or fusing representations does not improve the student model's performance under varying noise.

Therefore, we propose a multi-teacher knowledge distillation approach to address the limitations of variation-aware training and ensemble learning, and to mitigate accuracy uncertainty arising from non-idealities in the CIM architecture. We use the parameter α to give greater weight to samples in which the teacher network is more confident but the student network performs poorly, and vice versa to weaken the importance of these samples.

$$\mathcal{L}_{KD}(\hat{\boldsymbol{y}}^t, \hat{\boldsymbol{y}}^s) = \alpha D_{KL}(\beta \hat{\boldsymbol{y}}^t, \hat{\boldsymbol{y}}^s), \quad \text{where} \quad \alpha = (1 - \exp(-\frac{\log \hat{y}_i^s}{\log \hat{y}_i^t})). \tag{4}$$

We employed two parameters to regulate the knowledge transferred from the teacher models to the student model, enabling it to handle varying noise levels. The clean teacher model and the multi-teacher ensemble emphasize different aspects of knowledge distillation. Thus, we introduced the parameter α to balance their contributions. Additionally, the multi-teacher models, each specialized for distinct noise conditions β (explained in the manuscript), provide weighted coefficients that guide the student model in effectively adapting to diverse noise scenarios during inference.

1.5 Supplementary Note 5: Benchmark datasets introduction

1.5.1 CIFAR-10 dataset

The CIFAR-10 dataset [17] is a widely used benchmark for image classification tasks. It consists of 60,000 32×32 color images distributed across 10 mutually exclusive classes, with 6,000 images per class. The dataset is pre-divided into a training set of 50,000 images and a test set of 10,000 images. The classes are completely balanced in both the training and test sets and represent common objects such as airplanes, automobiles, birds, and cats.

1.5.2 CIFAR-100 dataset

The CIFAR-100 dataset [17], akin to CIFAR-10, comprises 60,000 32×32 color images distributed across 100 fine-grained classes. Each class includes 600 images, divided into 500 for training and 100 for testing. A distinctive aspect of CIFAR-100 is its hierarchical organization, where the 100 classes are grouped into 20 superclasses, each encompassing 5 fine-grained classes. This arrangement facilitates the assessment of both fine-grained and coarse-grained classification performance.

1.5.3 Carvana dataset

The Carvana dataset [18] was released for the *Carvana Image Masking Challenge* on the Kaggle platform. It is specifically designed for high-precision image segmentation. The dataset contains high-resolution images of automobiles taken from various angles. Each image is paired with a corresponding high-quality, pixel-perfect binary mask that segments the car from the background. As specified in our main text, the dataset contains 5,088 images, which we partitioned into a training set of 4,700 images and a test set of 318 images for our experiments.

1.5.4 Berkeley segmentation dataset

The Berkeley segmentation dataset [19] (BSDS), specifically its variants like BSDS300 and BSDS500, is a benchmark collection of natural images originally created for evaluating image segmentation and boundary detection algorithms. In the context of image denoising, it is standard practice to use the high-quality, clean images from this dataset as ground truth. For training denoising models, noisy image pairs are generated by adding synthetic noise (e.g., Additive White Gaussian Noise) to these clean images. Our work follows this convention, utilizing 400 images from this dataset for generating training data.

1.5.5 Set12 dataset

Set12 [20] is a standard and widely recognized benchmark dataset used exclusively for testing the performance of image denoising algorithms. It consists of 12 commonly used grayscale images that cover a range of content, from detailed patterns to smooth regions. Due to its small size, it is not used for training but serves as a consistent test bed for quantitative performance comparison (e.g., using PSNR) across different denoising methods in the literature.

2 Supplementary Methods

2.1 Supplementary Methods 1: Impact of device non-idealities on DNNs' performance

To assess the impact of device non-idealities on deep neural networks' (DNNs) performance, we apply the two proposed noise models to PyTorch-based DNNs. The log-normal multiplicative noise model was evaluated with σ ranging from 0.1 to 0.5, while the perceptual normal additive noise model was tested with η between 0.02 and 0.1. First, we examined the impact of these noise models on DNN weights. We evaluated the impact of these noise models on three tasks: image classification, object segmentation, and image denoising. For image classification (Fig.1 a), we trained a VGG16 model on the CIFAR-10 dataset

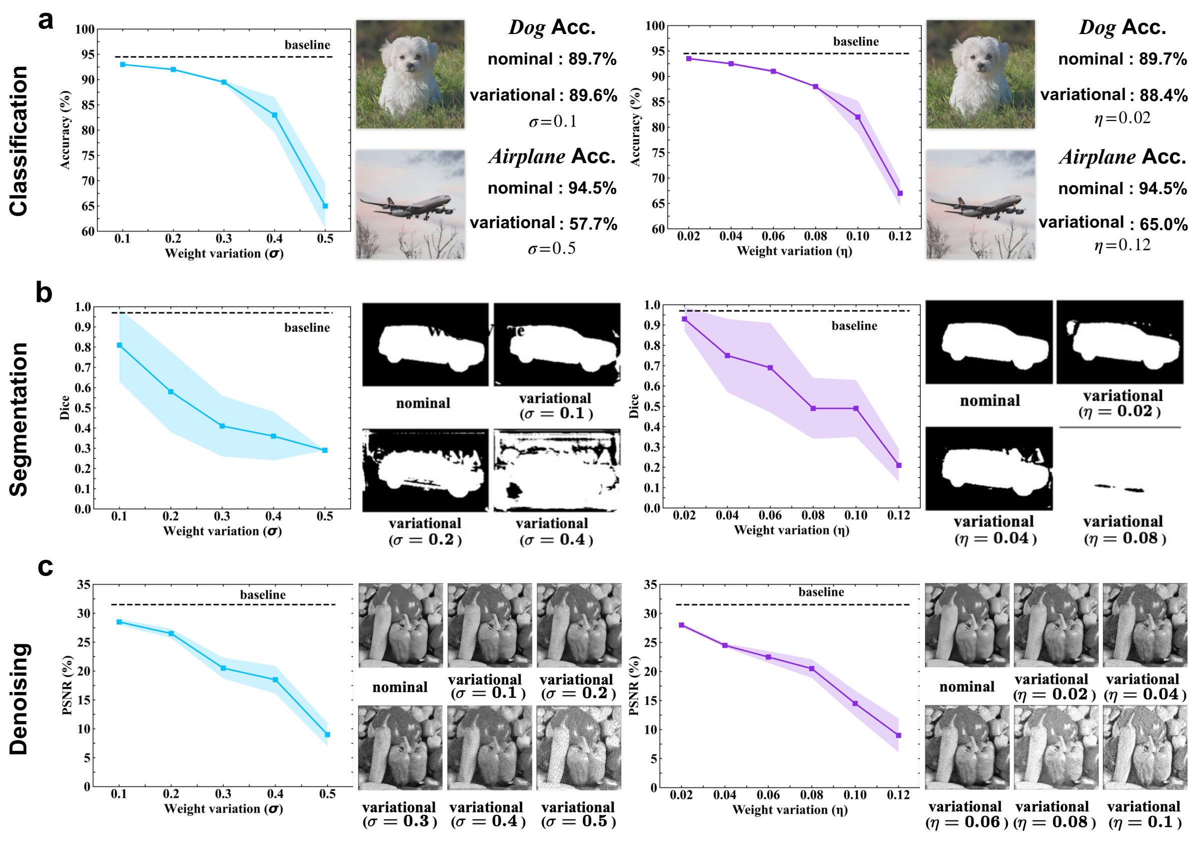

Figure 1: **a** Impact of non-idealities on image classification accuracy. The VGG16 model trained on CIFAR-10 exhibits a gradual decline in accuracy as noise intensity increases. **b** Effect of non-idealities on object segmentation performance. A UNet model trained on the Carvana dataset experiences a significant drop in Dice score with increasing noise levels, with segmentation capability nearly lost beyond $\sigma > 0.3$ or $\eta > 0.06$. **c** Impact of non-idealities on image denoising performance. The DnCNN model trained on the Berkeley dataset shows high sensitivity to weight perturbations, leading to severe degradation in denoising capability at higher noise levels.

and applied independent noise to each layer's weights. Under ideal conditions, the model achieved a 93.18% classification accuracy. However, under severe noise conditions, accuracy dropped significantly to 66.27% for log-normal multiplicative noise and 66.84% for perceptual normal additive noise. For object segmentation (Fig.1 b), a UNet model trained on the Carvana dataset exhibited a similar decline in performance. Notably, segmentation networks are more sensitive to weight perturbations compared to classification tasks due to their finer granularity. As noise levels increase, the Dice score drops sharply, and visualization results indicate that the model's segmentation capability is nearly completely lost when $\sigma > 0.3$ or $\eta > 0.06$. For image denoising (Fig.1 c), a DnCNN model trained on the Berkeley dataset demonstrated high sensitivity to weight noise. Even at $\sigma = 0.3$ or $\eta = 0.06$, the model's denoising capability was severely compromised, evidently leading to substantial image degradation. These results underscore the profound impact of device non-idealities on DNN performance, particularly in tasks requiring high weight precision. They highlight the critical need for robust training methodologies to mitigate the effects of hardware-induced variations.

2.2 Supplementary Methods 2: Impact of device non-idealities on DNN Weight

As shown in Fig.2, the blue distribution represents the nominal weights, whereas the yellow distribution shows the weights after noise injection. As noise levels increase, the variance of both distributions expands significantly, demonstrating the detrimental impact of device non-idealities on weight precision.

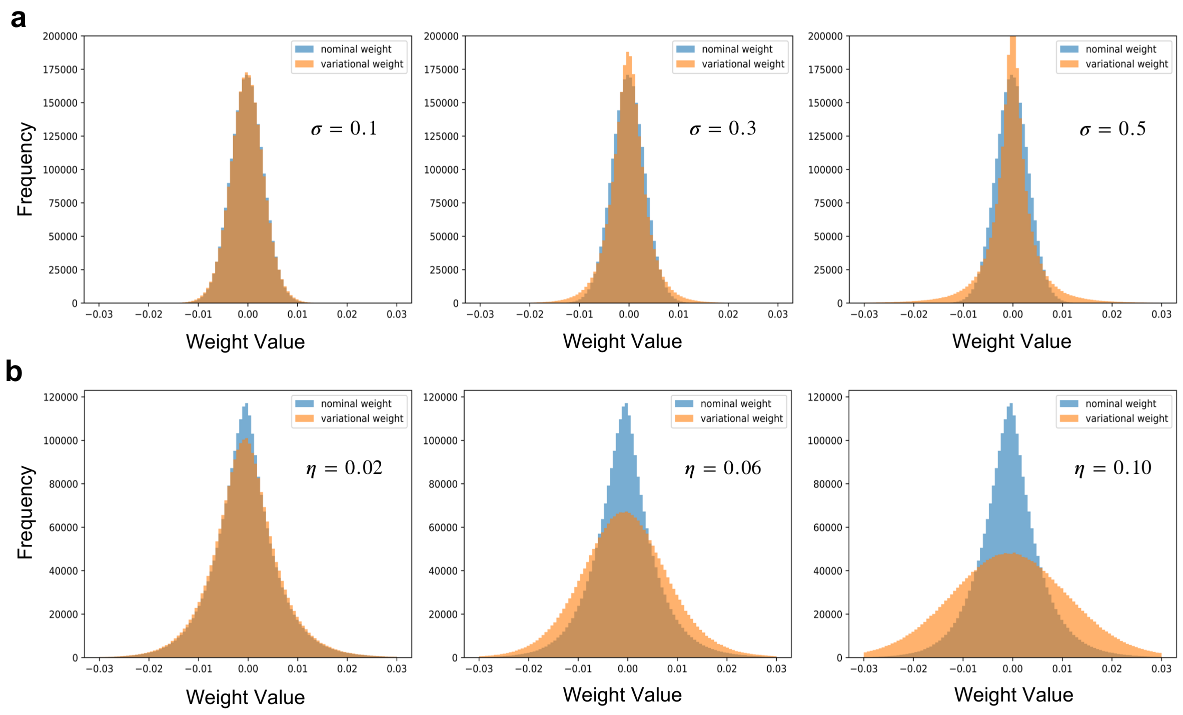

Figure 2: **a,b** Influence of log-normal multiplicative and perceptual normal additive noise on DNN weights. The blue distribution represents nominal weights, while the yellow distribution shows the weights after noise injection.

2.3 Supplementary Methods 3: Existing optimization methods for addressing device non-idealities

Two widely used algorithmic approaches—variation-aware training 1.2 and ensemble learning 1.3—aim to mitigate device non-idealities in CIM systems. Variation-aware training incorporates device variations into the training process, enhancing robustness against specific non-idealities. Ensemble learning, on the other hand, improves error tolerance by aggregating predictions from multiple independently trained models.

Limitations of Variation-Aware Training. Fig.3 a and c present the performance of the VGG16 model under log-normal multiplicative noise and perceptual Gaussian additive noise. To enable variation-aware training in these scenarios, we train separate models for each specific noise signature, as this approach requires integrating hardware variations (modeled as random variables or injected noise) directly into the training process. The model achieves optimal performance when the noise profile used during training closely matches the actual noise present during inference. Conversely, a mismatch between training and inference noise levels leads to significant performance degradation, indicating limited generalizability. Fig.3 b and d further demonstrate model stability under uncertain non-ideal conditions. The method exhibits greater stability when the training noise level is set to a median value ($\sigma = 0.3$ and $\eta = 0.06$), but the comprehensive performance is still far from the ideal value.

Limitations of Ensemble Learning. Fig.3 e–h display the results of ensemble learning, where multiple VGG16 models trained with different initializations are combined. To implement ensemble learning in these scenarios, we ensembled one to five models using weighted averaging to test the effectiveness of the ensemble model in dealing with log-normal multiplicative noise and perceptual normal additive noise. Fig.3 e and g illustrate test results of log-normal multiplicative noise and perceptual normal additive noise. While ensemble learning enhances performance for minor weight perturbations, its effectiveness diminishes under significant non-idealities. Fig.3 f and h further illustrate the method's performance under uncertain non-ideal errors. Although ensemble learning can improve model stability

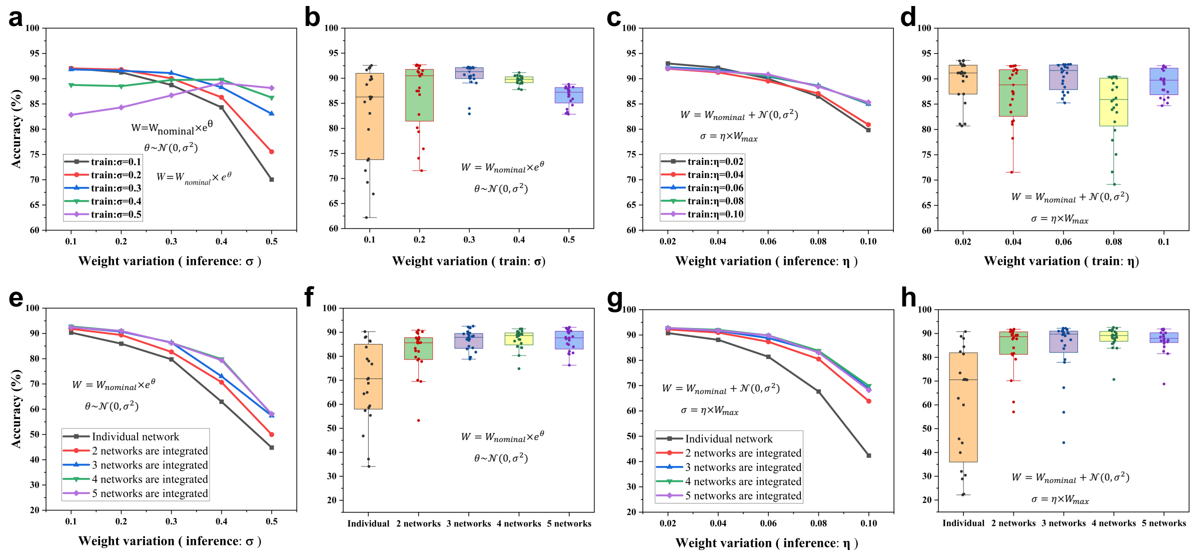

Figure 3: **a–d** Performance of variation-aware training. **a** Inference performance of a VGG16 model specifically trained with log-normal multiplicative noise under predefined weight variation levels. **b** Inference performance of a VGG16 model notably trained with log-normal multiplicative noise under randomly varying weight levels. **c** Inference performance of a VGG16 model, particularly trained with perceptual normal additive noise under predefined weight variation levels. **d** Inference performance of a VGG16 model effectively trained with perceptual normal additive noise under randomly varying weight levels. **e–h** Results of ensemble learning. **e** Inference performance of multiple VGG16 models under strictly predefined weight variation levels of log-normal multiplicative noise. **f** Inference performance of multiple VGG16 models under randomly varying weight levels of log-normal multiplicative noise. **g** Inference performance of multiple VGG16 models under predefined weight variation levels of perceptual normal additive noise. **h** Inference performance of multiple VGG16 models under randomly varying weight levels of perceptual normal additive noise.

in some cases, it fails to fully compensate for large-scale weight variations. Moreover, the increased computational cost of training and inference for multiple models makes ensemble learning less practical for resource-constrained CIM systems. While both variation-aware training and ensemble learning provide partial solutions to device non-idealities, they each come with notable critical drawbacks. Variation-aware training is limited by the requirement for accurate device variation characterization and particularly suffers from performance degradation when the training and deployment conditions differ. Ensemble learning offers improved robustness against minor variations but struggles with significant weight deviations and substantially increases computational complexity. These limitations underscore the need for a more scalable, adaptable, and generalizable innovative approach to mitigating device non-idealities in inherently CIM-based deep learning systems.

2.4 Supplementary Methods 4: Real hardware device-1T1R RRAM setup

This section details the construction process of the 1T1R RRAM test chip and the design of the custom hardware system used for its experimental characterization. To collect RRAM statistics, fully integrated nanoscale CMOS/RRAM structures were implemented on a 2.9×2.9 mm chip platform (an optical microscope image of which is shown in Fig.4 b) using China Semiconductor Manufacturing Corporation's (CSMC) 180nm CMOS process technology. The RRAM device stack consists of a 60 nm TiAuPd bottom electrode, a 6 nm hafnium oxide (HfO_2) memristive switching layer, and a 140 nm TaPdAu top electrode. This stack is integrated on the Via5 layers using a BEOL-compatible process flow, as shown in Fig.4 a. The structure was fabricated through processes such as photolithography, physical vapor deposition (PVD), atomic layer deposition (ALD), and gas source etching (GSE).

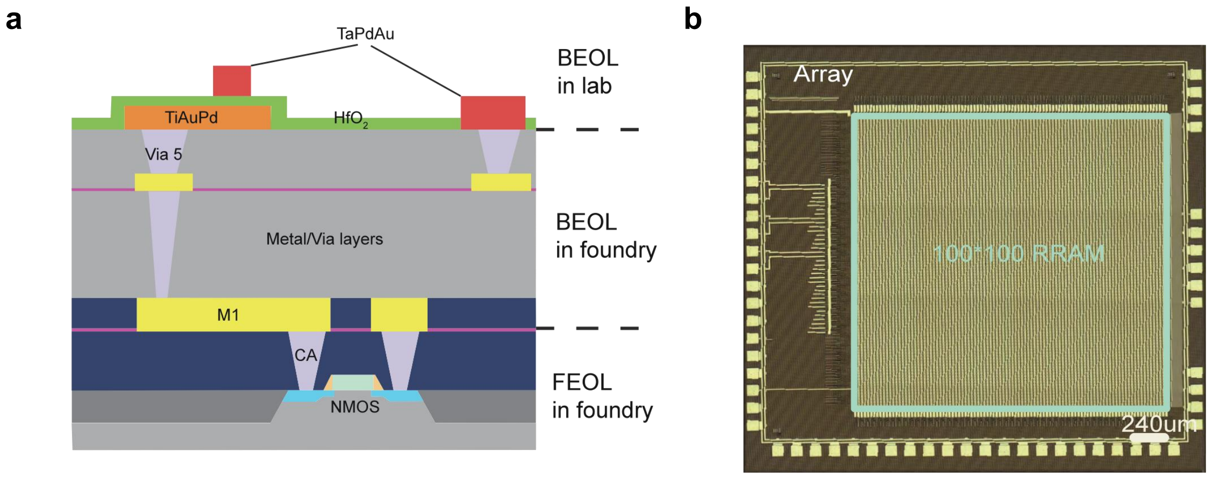

Figure 4: **a** RRAM schematic. **b** Optical microscope image of HfO_2 RRAM devices integrated with CMOS.

For memristor programming, a pulse-based approach was employed to apply pulses in the order of hundreds of nanoseconds for the set/reset operations of RRAM devices. A read pulse of 0.1 V with a 320 ns pulse width was applied after each set/reset operation to read out the high resistance state (HRS) and low resistance state (LRS). The maximum current during forming/set operations was controlled by modulating the gate voltage of the on-chip integrated NMOS devices. The 1T1R configuration allows for tighter current control, minimizing the effect of parasitic capacitances during switching. During the set operation on HfO_2 memristors, the primary method to achieve a lower resistance state is through current limiting, mainly realized by voltage-driven resistance change. The key parameter for the set operation is the gate voltage of the NMOS, which can be adjusted to regulate the magnitude of current limiting. In contrast, during the reset operation, the main mechanism is the thermal effect that melts the conductive filament within the memristor. The key parameter for the reset operation is the voltage difference between the bottom and top electrodes of the memristor.

To enable access to and programming of a 1T1R device, we developed a custom printed circuit board. The board features a digital-to-analog converter (DAC), an analog-to-digital converter (ADC), a microcontroller unit (MCU), and an instrumentation amplifier, with the PC acting as the host to coordinate tasks among the MCU and the 1T1R chip. The DAC provides the programming and read-out voltages, while the ADC and instrumentation amplifier sample and read out the current values to facilitate feedback-based programming. Based on the HfO_2 RRAM programming logic and testing hardware we previously described, we established a comprehensive testing system. Using this system, we characterized the fundamental properties and variations of memristors in real-world and complex scenarios.

2.5 Supplementary Methods 5: Pseudocode for image classification

To simulate the impact of device non-idealities on classification performance in image classification networks, we apply multiplicative exponential noise to the weights of convolutional and fully connected layers. Additive noise adopts a comparable approach, with the primary difference being the mode of noise integration. As shown in Algorithm 1, the pseudocode processes an input tensor with a noise parameter, first passing it through 13 convolutional layers where Gaussian noise is sampled and exponentiated to multiply the cloned weights. This is followed by convolution, batch normalization, and ReLU activation. Max pooling is added after specific layers. Global average pooling and flattening precede three fully connected layers with analogous noise injection, batch normalization, and ReLU.

Algorithm 1 Pseudocode for non-idealities in image classification networks

```
Input: Input tensor $x$, noise parameter $\lambda$
Output: Output tensor $x$
 1: // Convolutional layers with exponential noise
 2: for $i = 1$ to 13 do
 3:     Apply Conv2D with exponential noise:
 4:         $W_i \leftarrow$ clone (conv$_i$.weight)
 5:         noise$_i \sim \mathcal{N}(0, \lambda^2)$ with same size as $W_i$
 6:         $x \leftarrow$ Conv2D$(x, \exp(\text{noise}_i) \odot W_i, \text{bias}_i)$
 7:         $x \leftarrow$ BatchNorm$(x)$
 8:         $x \leftarrow$ ReLU$(x)$
 9:     if $i \in \{2, 4, 7, 10\}$ then
10:             $x \leftarrow$ MaxPool2D$(x)$
11:     end if
12: end for
13: // Global average pooling
14: $x \leftarrow$ AdaptiveAvgPool2D$(x)$
15: $x \leftarrow$ Flatten$(x)$
16: // Fully connected layers with exponential noise
17: for $j = 1$ to 3 do
18:     Apply Linear with exponential noise:
19:         $W_{\text{linear}_j} \leftarrow$ clone(linear$_j$.weight)
20:         l_noise$_j \sim \mathcal{N}(0, \lambda^2)$ with same size as $W_{\text{linear}_j}$
21:         $x \leftarrow$ Linear$(x, \exp(\text{l\_noise}_j) \odot W_{\text{linear}_j}, \text{bias}_j)$
22:     if $j < 3$ then
23:             $x \leftarrow$ BatchNorm$(x)$
24:             $x \leftarrow$ ReLU$(x)$
25:     end if
26: end for
27: return $x$
```

2.6 Supplementary Methods 6: Pseudocode for image segmentation

To simulate the impact of device non-idealities on segmentation performance in image segmentation networks, we apply additive Gaussian noise to the weights of convolutional layers in the encoder, bottleneck, decoder, and final convolution. The noise variance is scaled by the maximum weight value and the noise parameter. Multiplicative noise follows a similar procedure, differing only in the manner of noise incorporation. As shown in the pseudocode algorithm 2 and 3, which process an input tensor through the encoder and decoder using convolutional blocks. In these blocks, weights are cloned, noise is sampled from a Gaussian distribution with standard deviation proportional to the maximum weight, and added to the weights prior to convolution, followed by batch normalization and ReLU activation. The final convolution applies analogous noise without subsequent normalization or activation.

Algorithm 2 Pseudocode for non-idealities in image segmentation networks

```
Input: Input tensor $x$, noise parameter $\lambda$
Output: Output tensor $x$
 1: // Encoder path with noise
 2: for each Conv non-idealities block in encoder do
 3:     Apply non-idealities with noise:
 4:         $W \leftarrow$ clone(conv.weight)
 5:         $\sigma \leftarrow \lambda \times \max(W)$
 6:         noise $\sim \mathcal{N}(0, \sigma^2)$
 7:         $x \leftarrow$ Conv2D$(x, W +$ noise$)$
 8: end for
 9: // Bottleneck with noise
10: // Decoder with noise
11: for each Conv_nonid block in decoder do
12:     Apply same noise mechanism as Encoder
13: end for
14: // Final convolution with noise
15: $W_{\text{final}} \leftarrow$ clone(Conv$_{\text{final}}$.weight)
16: $\sigma \leftarrow \lambda \times \max(W_{\text{final}})$
17: noise$_{\text{final}} \sim \mathcal{N}(0, \sigma^2)$
18: $x \leftarrow$ Conv2D$(x, W_{\text{final}} +$ noise$_{\text{final}})$
19: return $x$
```

Algorithm 3 Conv add non-idealities forward bolck

```
Input: Input tensor $x$, noise parameter $\lambda$
Output: Output tensor $x$
    // Each convolution with noise
 1: $W \leftarrow$ clone(conv.weight)
 2: max_value $\leftarrow \max(W)$
 3: $\sigma \leftarrow \lambda \times$ max_value
 4: noise $\sim \mathcal{N}(0, \sigma^2)$ with size of $W$
 5: $x \leftarrow$ Conv2D$(x, W +$ noise$)$
 6: $x \leftarrow$ BatchNorm$(x)$
 7: $x \leftarrow$ ReLU$(x)$
 8: return $x$
```

2.7 Supplementary Methods 7: Pseudocode for image denoising

To simulate the impact of device non-idealities on denoising performance, we apply multiplicative exponential noise to the weights of convolutional layers in image denoising networks. We perform the same additive noise, adding the noise to the convolutional layers. The pseudocode is shown in Algorithm 4. It processes an input tensor with a noise parameter through 17 convolutional layers. For each layer, Gaussian noise is sampled and exponentiated to multiply the weights prior to convolution. The first layer is followed by ReLU activation without batch normalization. Intermediate layers (2 to 16) include batch normalization and ReLU. The final layer applies only the convolution without subsequent normalization or activation.

Algorithm 4 Pseudocode for non-idealities in image denoising networks

```
Input: Input tensor $x$, noise parameter $\lambda$
Output: Output tensor $x$
1: // First convolution layer (no batch normalization)
2: $\text{noise}_1 \sim \mathcal{N}(0, \lambda^2)$ with size of conv1.weight
3: $x \leftarrow \text{Conv2D}(x, \exp(\text{noise}_1) \odot W_1)$
4: $x \leftarrow \text{ReLU}(x)$
5: // Intermediate convolution layers (2 to 16)
6: for $i = 2$ to $16$ do
7:     $\text{noise}_i \sim \mathcal{N}(0, \lambda^2)$ with size of conv$_i$.weight
8:     $x \leftarrow \text{Conv2D}(x, \exp(\text{noise}_i) \odot W_i)$
9:     $x \leftarrow \text{BatchNorm}(x)$
10:    $x \leftarrow \text{ReLU}(x)$
11: end for
12: // Final convolution layer (no batch normalization and ReLU)
13: $\text{noise}_{17} \sim \mathcal{N}(0, \lambda^2)$ with size of conv17.weight
14: $x \leftarrow \text{Conv2D}(out, \exp(\text{noise}_{17}) \odot W_{17})$
15: return $x$
```

3 Supplementary Figure

3.1 Supplementary Figure 5: Illustration of CIM architecture and analysis of weight distortion induced by ReRAM and PCM device conversions

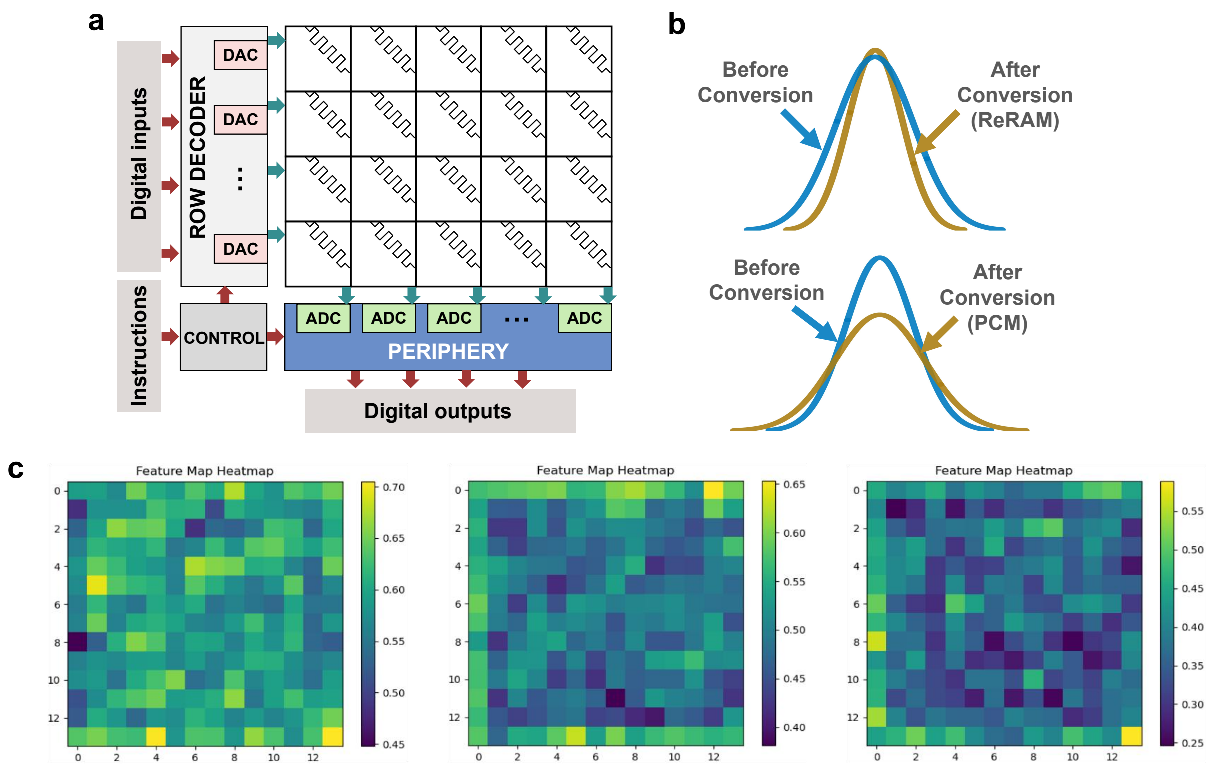

Figure 5: **a** Schematic of the computing-in-memory (CIM) architecture. The diagram illustrates a crossbar array integrated with digital-to-analog converters (DACs), analog-to-digital converters (ADCs), and peripheral circuits that manage digital input and output. **b** Comparison of weight distributions before and after memory device conversion. The left panel illustrates changes following resistive random-access memory (ReRAM) conversion, and the right panel shows changes due to phase-change memory (PCM) conversion. In both cases, the distribution curves reflect the impact of analog non-idealities on synaptic weight representation. **c** Visualization of the weight matrices before and after conversion. The left plot shows the original matrix, the middle plot shows the matrix after ReRAM conversion, and the right plot shows the result after PCM conversion. These visualizations emphasize the distortions and noise introduced by different memory technologies within the CIM system.

3.2 Supplementary Figure 6: Noise-aware weight perturbation and quantization performance evaluation on CIM simulators

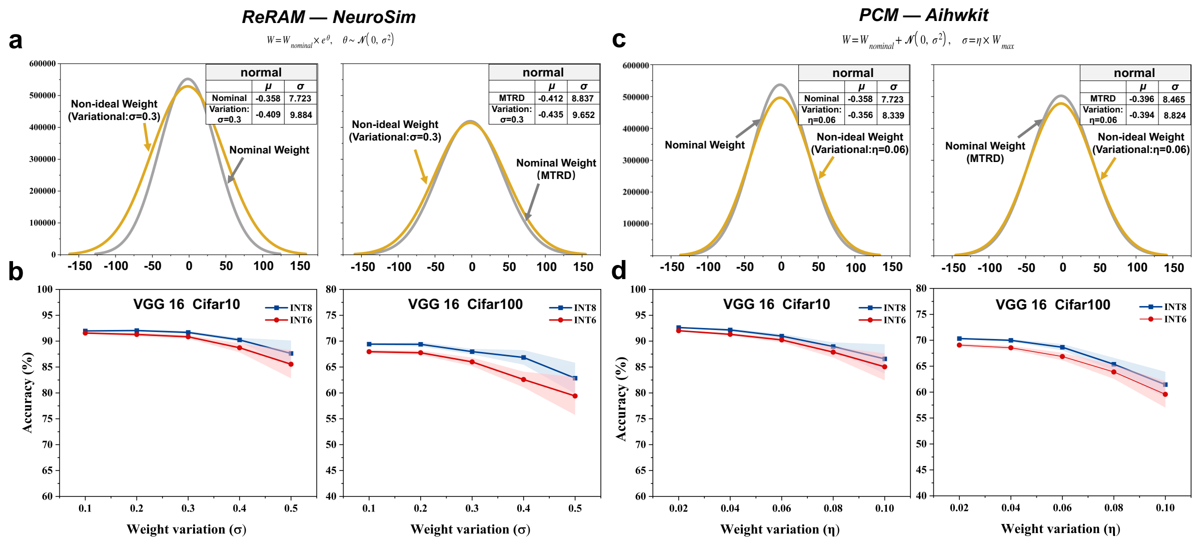

Figure 6: **a** Weight distribution modeling of RRAM-based analog CIM using the NeuroSim simulator. The left and right figures show the weight distribution differences under multiplicative noise for the standard and proposed MTRD models, respectively. The MTRD model exhibits reduced weight fluctuation under noise, indicating improved stability. **b** Image classification accuracy of the VGG16 model on CIFAR-10 and CIFAR-100 datasets simulated with NeuroSim. Accuracy degradation is shown as a function of multiplicative noise parameter variation under INT8 and INT6 quantization. **c** Weight distribution modeling of PCM-based analog CIM using the Aihwkit simulator. The left and right figures display weight distribution differences under additive noise for the standard and proposed MTRD models, respectively. The MTRD model shows reduced weight fluctuation under noise, demonstrating enhanced stability. **d** Image classification accuracy of the VGG16 model on CIFAR-10 and CIFAR-100 datasets simulated using Aihwkit. Accuracy degradation versus additive noise parameter for INT8 and INT6 quantization.

3.3 Supplementary Figure 7: Endurance statistical analysis

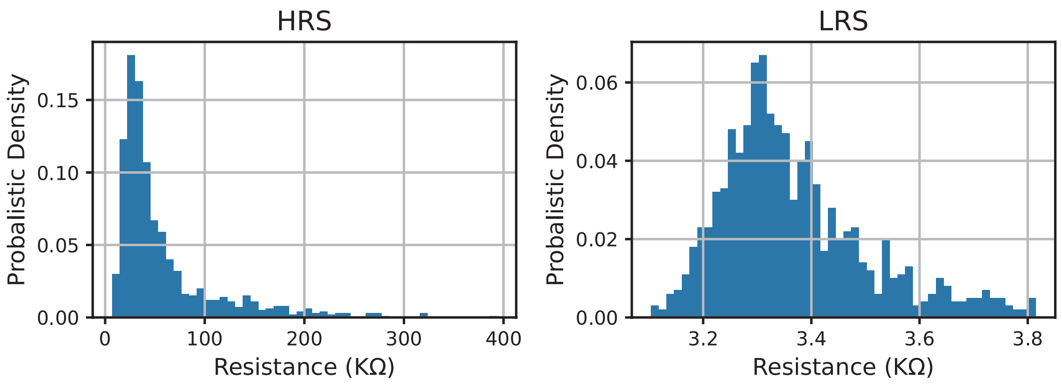

Figure 7: Illustration of the variation of resistance state with the same operating parameter.

3.4 Supplementary Figure 8: Retention statistical analysis

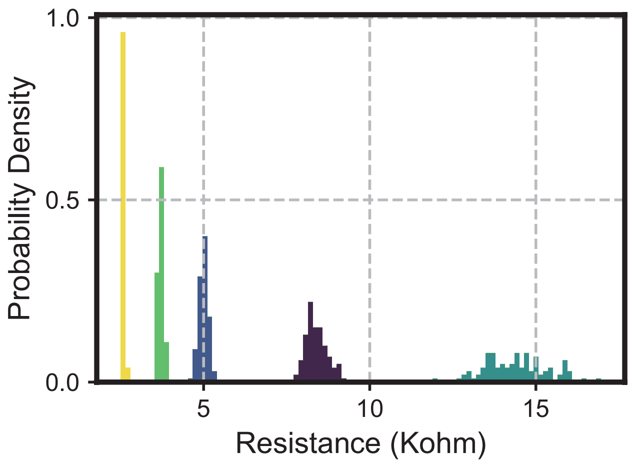

Figure 8: Illustration of the variation of resistance states over time.

3.5 Supplementary Figure 9: Schematic of image denoising under multiplicative noise perturbation

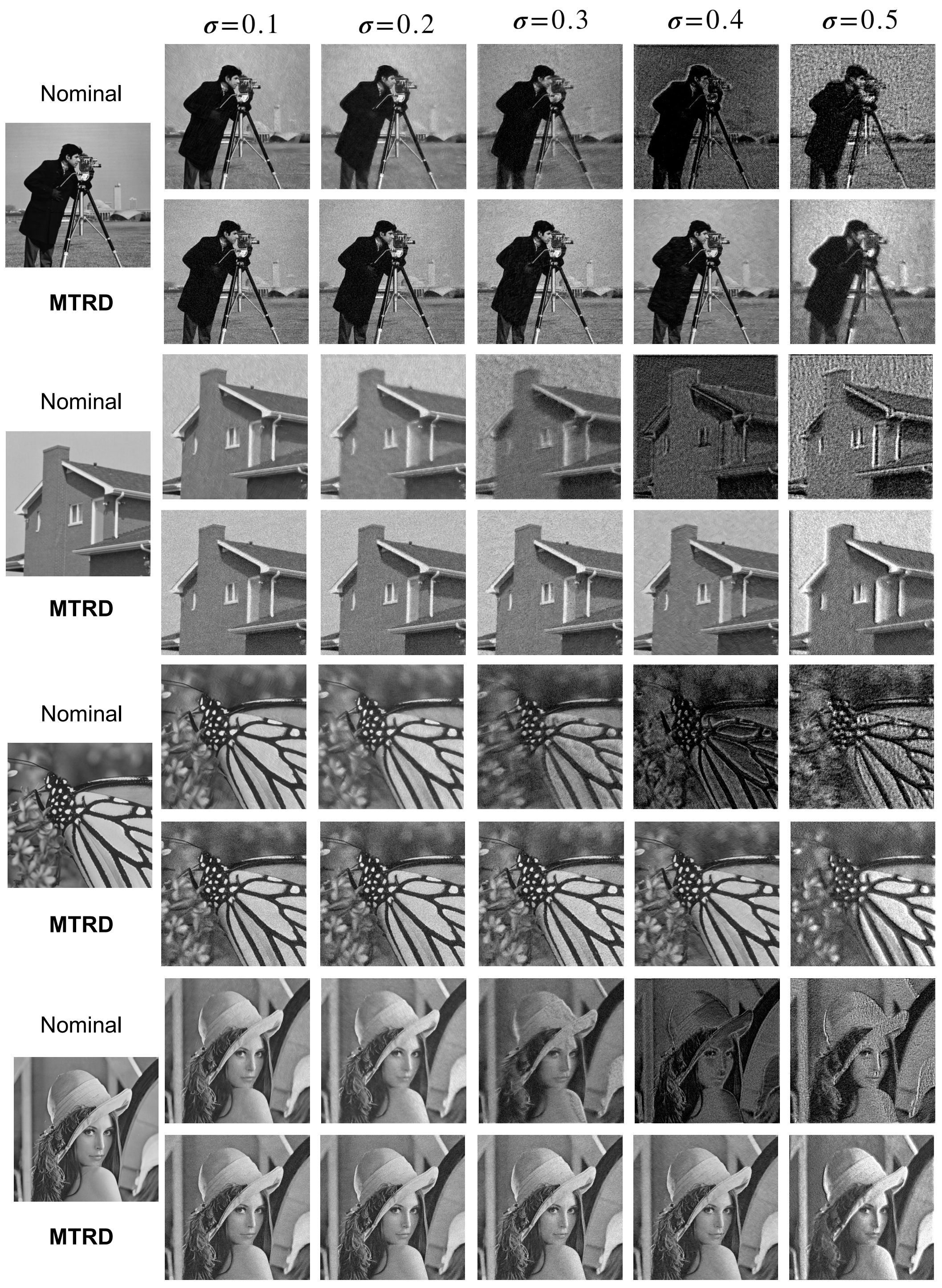

Figure 9: Illustration of image denoising performance using the nominal and MTRD methods under varying levels of multiplicative noise.

3.6 Supplementary Figure 10: Schematic of image denoising under additive noise perturbation

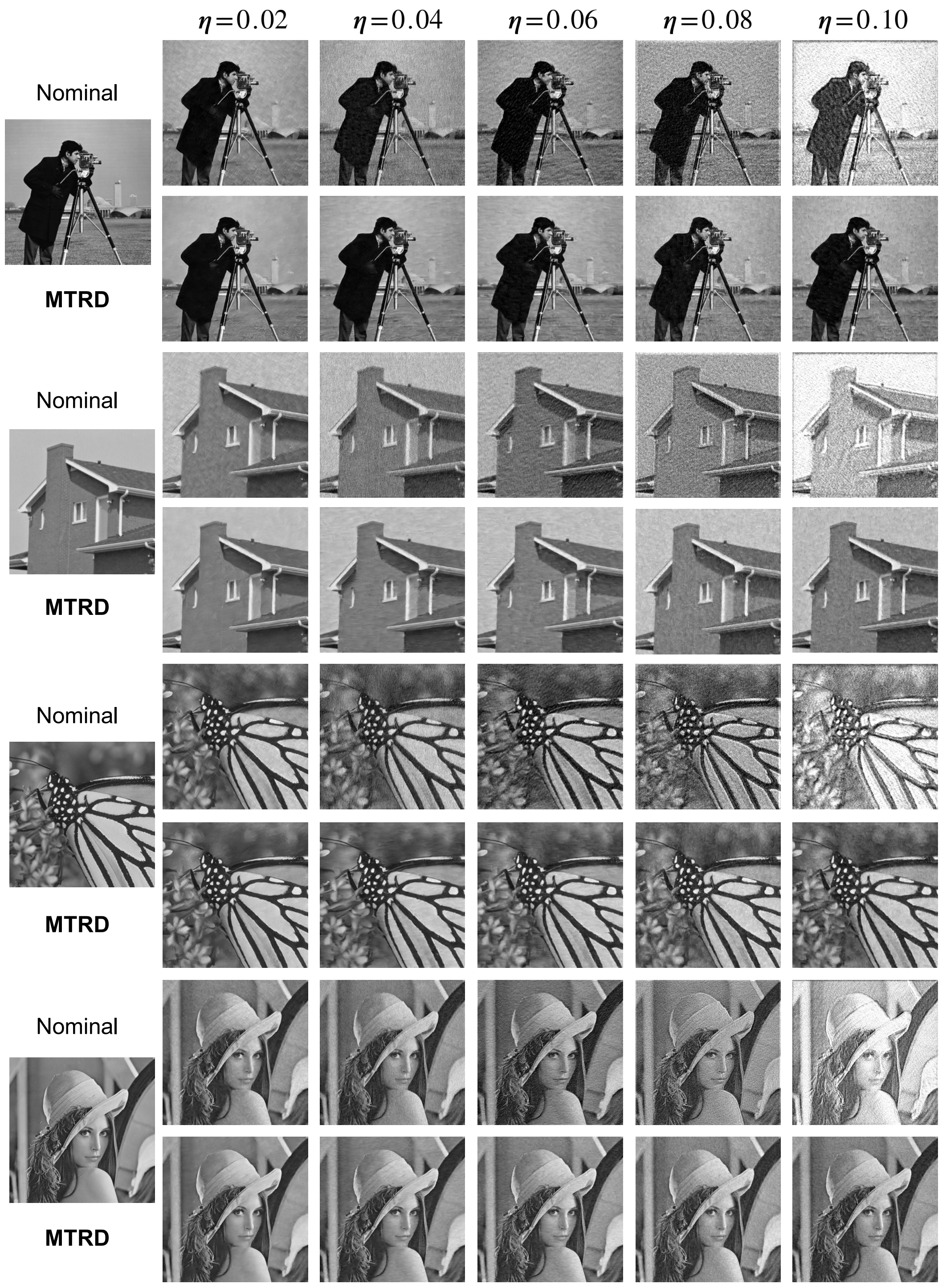

Figure 10: Illustration of image denoising performance using the nominal and MTRD methods under varying levels of additive noise.

References

[1] Weier Wan, Rajkumar Kubendran, Clemens Schaefer, Sukru Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen Deiss, Priyanka Raina, He Qian, Bin Gao, et al. A compute-in-memory chip based on resistive random-access memory. *Nature*, 608(7923):504–512, 2022.

[2] Shaocong Wang, Yizhao Gao, Yi Li, Woyu Zhang, Yifei Yu, Bo Wang, Ning Lin, Hegan Chen, Yue Zhang, Yang Jiang, et al. Random resistive memory-based deep extreme point learning machine for unified visual processing. *Nature Communications*, 16(1):960, 2025.

[3] Yudeng Lin, Huaqiang Wu, Bin Gao, Peng Yao, Wei Wu, Qingtian Zhang, Xiang Zhang, Xinyi Li, Fuhai Li, Jiwu Lu, et al. Demonstration of generative adversarial network by intrinsic random noises of analog rram devices. In *IEEE International Electron Devices Meeting (IEDM)*, pages 3–4. IEEE, 2018.

[4] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. Vortex: Variation-aware training for memristor x-bar. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[5] Ying Zhu, Grace Li Zhang, Tianchen Wang, Bing Li, Yiyu Shi, Tsung-Yi Ho, and Ulf Schlichtmann. Statistical training for neuromorphic computing using memristor-based crossbars considering process variations and noise. In *Design, Automation and Test in Europe*, 2020.

[6] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. Design of reliable dnn accelerator with unreliable reram. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1769–1774. IEEE, 2019.

[7] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 19–24. IEEE, 2017.

[8] Dovydas Joksas, Pedro Freitas, Zheng Chai, Wing H Ng, Mark Buckwell, C Li, WD Zhang, Q Xia, AJ Kenyon, and A Mehonic. Committee machines—a universal method to deal with non-idealities in memristor-based neural networks. *Nature communications*, 11(1):4273, 2020.

[9] Tao Liu, Wujie Wen, Lei Jiang, Yanzhi Wang, Chengmo Yang, and Gang Quan. A fault-tolerant neural network architecture. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[10] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 558–567, 2019.

[11] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1285–1294, 2017.

[12] Zhiqiang Shen and Marios Savvides. Meal v2: Boosting vanilla resnet-50 to 80%+ top-1 accuracy on imagenet without tricks. *arXiv preprint arXiv:2009.08453*, 2020.

[13] Liuyu Xiang, Guiguang Ding, and Jungong Han. Learning from multiple experts: Self-paced knowledge distillation for long-tailed classification. In *European conference on computer vision*, pages 247–263. Springer, 2020.

[14] Ruifei He, Shuyang Sun, Jihan Yang, Song Bai, and Xiaojuan Qi. Knowledge distillation as efficient pre-training: Faster convergence, higher data-efficiency, and better transferability. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9161–9171, 2022.

[15] SeongUk Park and Nojun Kwak. Feed: Feature-level ensemble for knowledge distillation. *arXiv preprint arXiv:1909.10754*, 2019.

[16] Iou-Jen Liu, Jian Peng, and Alexander G Schwing. Knowledge flow: Improve upon your teachers. *arXiv preprint arXiv:1904.05878*, 2019.

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[18] Brian Shaler, DanGill, Maggie, Mark McDonald, Patricia, and Will Cukierski. Carvana image masking challenge, 2017.

[19] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings eighth IEEE international conference on computer vision*, volume 2, pages 416–423. IEEE, 2001.

[20] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.